НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 123 «Комп'ютерна інженерія»

на тему: «Алгоритм балансування трафіку в інтелектуальних транспортних системах»

Виконав:
студент IV курсу, групи ІО-62
Шапка Олексій Володимирович _____

Керівник:
Доцент, кандидат технічних наук,
Павлов Валерій Георгійович _____

Консультант з нормоконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
Посада, науковий ступінь, вчене звання,
Прізвище, ім'я, по батькові _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Шапці Олексію Володимировичу

1. Тема проєкту «Алгоритм балансування трафіку в інтелектуальних транспортних системах», керівник проєкту Павлов Валерій Георгійович, доцент, к.т.н., затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту 06 червня 2020 р.

3. Вихідні дані до проєкту: технічне завдання, науково-технічна література

4. Зміст пояснювальної записки: порівняльний аналіз існуючих програмних рішень, вибір засобів реалізації та опис отриманої системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) :

1. Функціональна схема – плакат
2. Принципова схема – плакат
3. Структурна схема – плакат

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В.П., професор		

7. Дата видачі завдання 01 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	01.09.2019-22.12.2019	
2	Вивчення та аналіз завдання	23.12.2019-20.03.2020	
3	Розробка архітектури та загальної структури системи	20.03.2020-01.04.2020	
4	Розробка структур окремих підсистем	01.04.2020-10.04.2020	
5	Програмна реалізація системи	11.04.2020-20.04.2020	
6	Оформлення пояснювальної записки	01.05.2020-23.05.2020	
7	Передзахист	24.05.2020-26.05.2020	
8	Захист	15.06.2020-20.06.2020	

Студент Олексій ШАПКА

Керівник Валерій ПАВЛОВ

АНОТАЦІЯ

Дана дипломна робота присвячена розробці алгоритму, який рівномірно розподіляє навантаження для інтелектуальної транспортної системи представленої у вигляді планарного графу.

За допомогою інтерфейсу користувач може додавати чи видаляти вершини, редагувати їх зв’язки, встановлювати дистанцію чи змінювати початкове навантаження на обраному шляху. А також навантажувати систему та отримувати деталі розподілу навантаження після його збільшення.

Для реалізації програмного продукту використовується мова програмування Python.

ANNOTATION

This thesis is devoted to the development of an algorithm that evenly distributes the load for the intelligent transport system presented in the form of a planar graph.

Using the interface, user can add or remove nodes, edit their connections, set the distance or change the initial load on the selected path. In addition to load the system, and get the load distribution details after its increase.

The Python programming language is used to implement the software product.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	*ІАЛЦ. 467800.001* ВП	Відомість проєкту	1	
3	А4	*ІАЛЦ. 467800.002* ТЗ	Технічне завдання	3	
4	А4	*ІАЛЦ. 467800.003* ПЗ	Пояснювальна записка	54	
5	А4	*ІАЛЦ. 467800.004* Д1	Схема класів	1	
6	А4	*ІАЛЦ. 467800.005* Д2	Алгоритм роботи програми	1	
7	А4	*ІАЛЦ. 467800.006* Д3	Схема взаємодії с користувачем	1	
8	А4	*ІАЛЦ. 467800.007* Д4	Текст програми	17	

				ІАЛЦ. 467800.001 ВП	
	ПІБ	Підп.	Дата		
Розробн.	Шапка О.В.			Відомість дипломного проєкту	Лист 1 / Листів 1
Керівн.	Павлов В.Г.				
Консульт.					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-62
Н/контр.	Сімоненко В.П.				
Зав.каф.	Стіренко С.Г.				

Технічне завдання

до дипломного проєкту

на тему: «Алгоритм балансування трафіку в інтелектуальних транспортних системах»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку програмного додатку на тему «Алгоритм балансування трафіку в інтелектуальних транспортних системах».

Область застосування: рівномірний розподіл навантаження по інтелектуальній транспортній системі.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою роботи є створення інтелектуальної транспортної системи з вбудованим алгоритмом балансування трафіку, яка повинна оптимально розподіляти навантаження по системі та динамічно змінювати маршрути учасників транспортного руху відповідно до поточного стану навантаження транспортних вузлів.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є технічна документація для інтерфейсів прикладного програмування та бібліотек, що використовуються під час розробки програмного продукту, науково-технічна література з інформаційних технологій, публікації в періодичних виданнях, а також відповідні статті в мережі Інтернет за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- Застосування алгоритму пошуку оптимального шляху для балансування трафіку, який дозволить рівномірно розподілити навантаження по системі з урахуванням динамічного додання навантаження.
- Розробка інтерфейсу, який забезпечить взаємодію користувача з системою, за допомогою якого користувач повинен мати змогу завантажувати з файлу, створювати чи редагувати власну транспортну систему у вікні редактору, модифікувати її параметри, а саме відстань між вузлами та їх навантаження. Повинен бути передбачений функціонал для додавання навантаження на систему з відображенням процесу навантаження та вікна з детальною інформацією про додане навантаження.

5.2. Вимоги до програмного забезпечення

- Мова програмування Python.

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Core 2 / Athlon 64 і вище.
- Оперативної пам’яті не менше 4096 Мбайт.
- 100 Мбайт вільного місця на пристрої.

Пояснювальна записка

до дипломного проєкту

на тему: «Алгоритм балансування трафіку в інтелектуальних транспортних системах»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

GPS	(англ. Global Positioning System) — сукупність радіоелектронних засобів, що дозволяє визначати положення та швидкість руху об'єкта на поверхні Землі або в атмосфері.
ITC	Інтелектуальна транспортна система.
PRT	Персональний автоматичний транспорт (англ. Personal Rapid Transit) — це різновид міського та приміського транспорту, який автоматично перевозить пасажирів у режимі таксі, використовуючи мережу виділених шляхів.
IDA*	Ітераційне поглиблення алгоритму A* (англ. Iterative deepening A*).
IDDFS	Алгоритм пошуку в глибину з ітеративним поглибленням (англ. Iterative deepening depth-first search).
IDE	Інтегроване середовище розробки (англ. Integrated development environment) — це програма для розробки програмного забезпечення.
GUI	Графічний інтерфейс користувача (англ. GUI, Graphical user interface).

ВСТУП

Враховуючи досить швидкі темпи розвитку транспортної інфраструктури в сучасному світі зростає потреба в організації ефективної взаємодії між транспортними засобами. В даній роботі буде розглянуто алгоритм для організації оптимального розподілу навантаження в транспортній системі.

Актуальність теми

Зі збільшенням кількості транспортних засобів збільшується навантаження на транспортні вузли. Ця проблема особливо актуальна для міст з великою кількістю доріг та транспорту. Адже ефективне розподілення наземного транспорту -– це не лише питання особистого часу, а й ефективного функціонування міст.

Мета і задачі дослідження

Метою роботи є створення інтелектуальної транспортної системи з вбудованим алгоритмом балансування трафіку, яка повинна оптимально розподіляти навантаження по системі та динамічно змінювати маршрути учасників транспортного руху відповідно до поточного стану навантаження транспортних вузлів. Для досягнення цієї мети були поставлені наступні задачі:

- Розгляд основної інформації про транспортні системи
- Розгляд вже існуючих алгоритмів, їх аналіз, виокремлення переваг та недоліків;
- Розробка користувацького інтерфейсу для підтримки необхідного функціоналу;
- Створення функціоналу відповідно до поставленої задачі;
- Тестування створеного програмного продукту, опрацювання недоліків, покращення функціоналу та модернізація GUI.

Практичне значення

Результатом роботи повинна бути інтелектуальна транспортна система зі зручним та зрозумілим графічним інтерфейсом, за допомогою якого організовується проста взаємодія між користувачем та програмою. Шляхом моделювання демонструється ефективність використання алгоритму балансування. Дана програма має також відображати можливі майбутні проблеми при створенні нових транспортних вузлів на основі сконструйованої системи з заданими параметрами.

РОЗДІЛ 1
ОГЛЯД ТРАНСПОРТНИХ СИСТЕМ

1.1 Транспортна система. Загальні поняття

Транспортна система — це система пов’язаних складових (людей, задіяних в транспортному процесі; інфраструктури; транспортних засобів тощо), яка призначена для транспортування будь-чого (будь-кого). Вона означає державну транспортну інфраструктуру та пов'язані з нею системи, включаючи автомобільні дороги та платні дороги, відкриті для громадськості та пов'язані з ними права на проїзд, мости, транспортні засоби, обладнання, паркові та проїзні ділянки, транзитні станції, системи управління транспортом, інтелектуальний транспортний засіб системи автомобільних доріг та інші системи наземного транспорту.

Вони можуть включати в себе декілька транспортних вузлів с одним типом транспорту (райони міста, селища, тощо) або представляти собою комплекс взаємопов’язаних транспортних систем (міста, держави, континенти).

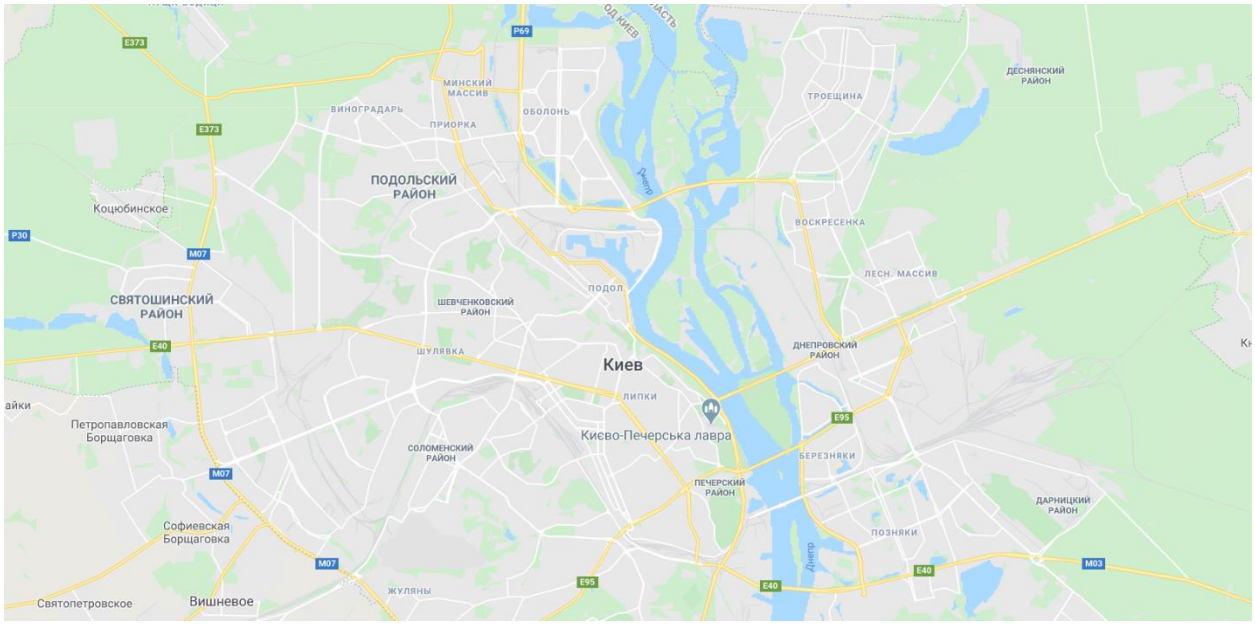


Рисунок 1.1 – Приклад транспортної системи міста мільйонника

Транспортні системи забезпечують інфраструктуру, необхідну для задоволення потреб суспільства. Зрештою, роль транспортної системи полягає у подоланні проблем, пов’язаних з фізичним розділенням між землекористуванням, товарами, послугами та людьми. Зростання попиту на подорожі протягом останніх десятиліть призвело до ряду значних проблем, пов'язаних з транспортною політикою. Головними серед них є зовнішні впливи, що виробляються транспортними системами. У цьому контексті шумове забруднення є однією з найактуальніших екологічних проблем, пов'язаних з транспортом. Це створює ключові проблеми для розробників політики, не в останню чергу, стосовно того, як слід оцінювати, контролювати та зменшувати рівень шуму від транспортних джерел у майбутньому. [1]

Транспорт – це одна з найважливіших складових виробничої інфраструктури. Його ефективне функціонування є необхідною умовою стабілізації, покращення структурних перетворень економіки, розвитку зовнішньоекономічних сфер діяльності, підвищення життєвого рівня населення, забезпечення національної безпеки країни.

Він належить до сфери матеріального виробництва, є його важливою галуззю і продовжує виробничий процес, доставляючи товари від місця виробництва до місця споживання. Продукцією транспорту є сам процес переміщення, який здійснюється за допомогою транспортних засобів як у сфері виробництва, так і у сфері обігу.

Також транспорт впливає на розвиток господарств як споживач ресурсів таких як метал, деревина, гума тощо. На нього припадає велика частина основного виробництва фондів та промислово-виробничого робітників.

Специфіка транспорту, як галузі господарства, полягає в тому, що він сам не виробляє продукцію, а бере учать у її створенні, забезпечує виробництво сировиною, матеріалами, обладнанням і перевозить готові вироби споживачу.

Він є важливою складовою частиною ринкової інфраструктури, бо створює умови для формування загальнодержавних та місцевих ринків.

Значення транспорту для будь-якої країни, що займає велику територію, неймовірно велике.

Економічна роль транспорту проявляється, перш за все, в тому, що він є невід'ємною ланкою кожного виробництва, і виконує неперервну і масову постановку безлічі видів сировини, палива, продукції з пунктів виробництва до споживача, поміж тим здійснює розподіл праці. Без транспорту неможливо оптимально розмістити виробництво. Транспорт це також важливий фактор економічної інтеграції країн та розвитку світової торгівлі.

Соціально-політичні функції транспорту проявляються в його можливості здійснювати обмін матеріальними цінностями між районами, містами, територіями і це сприяє їх об'єднанню в одну державу. Транспорт забезпечує вантажні, побутові чи туристичні поїздки, а також медичне обслуговування людей, та полегшує фізичну працю.

Культурне значення транспорту, важливе перш за все тим, що він забезпечує спілкування між континентами, країнами, містами, людьми, та сприяє задоволенню їх культурному потреб та обміну.

Оборонна роль транспорту в контексті держави завжди була важливою та актуальною. У всі часи транспорт розглядався як один з основних факторів забезпечення обороноздатності держави. З основних функцій можна виділити зміна позиції військ і озброєнь, забезпечення тилових об'єктів і військового промисловості.

Історія розвитку транспорту невід'ємна від історії людства. Причиною цього є потреба в переміщенні знарядь та предметів праці, без якої неможливі ні виробництво споживання, ні яка-небудь інша діяльність.

Транспорт неможливо переоцінити в розвитку процесу глобалізації. Завдяки справжньому прориву в транспортних технологіях, економічна глобалізація досягла таких успіхів. Контейнерна транспортна система стала основою інфраструктурного розвитку глобальної економіки. За декілька десятиліть глобалізації світова транспортна система отримала якісні зміни. З обслуговуючої галузі економіки, яка складалася з нез'єднаних видів

транспорту, система перетворилася на потужний комплекс з величезним рівнем внутрішньої інтеграції, характеристики якої багато в чому визначають напрямок розвитку світових виробничих ресурсів і споживчих ринків.

Транспортні системи створені в самих різних розмірах. Це можуть бути як місцеві транспортні мережі, що охоплюють автобусну мережу для міста та його околиць, так і міждержавні мережі доставки. Чим більше відстань, тим ефективнішим є використання транспортних засобів в транспортній системі.

Транспортні системи, незалежно від типу, можуть втрачати ефективність через затори. Затори трафіку впливають на транспортні мережі і виникають у міру збільшення використання. Більш тривалий час поїздки та менша швидкість - основні характеристики транспортних заторів. Коли попит наближається до потужності транспортного засобу, виникають надзвичайні затори. А так як транспортна галузь є ключовою у більшості аспектів людської життєдіяльності потрібні модернізації звичайних транспортних вузлів і переходу до інтелектуальних транспортних систем, які забезпечують ефективне використання транспортних шляхів. [2]

1.2 Інтелектуальні транспортні системи. Загальні поняття

Ідея розумного перетворення міст у цифрові це полегшення життя його громадян у будь-якому аспекті. Розумна транспортна система стає незамінним компонентом серед усіх. У будь-якому місті мобільність є ключовою проблемою; будь то школа, коледж та офіс, або для будь-якої іншої мети громадяни використовують транспортну систему для подорожі по місту. Залучення громадян з інтелектуальною транспортною системою може заощадити свій час та зробити місто ще розумнішим. Інтелектуальна транспортна система (ITC) має на меті досягти ефективності руху за рахунок мінімізації транспортних проблем. Вона збагачує користувачів попередньою інформацією про рух, місцевою зручністю інформації в режимі реального часу, наявністю сидінь тощо, що скорочує час подорожі пасажирів, а також підвищує їх безпеку та комфорт.

Інтелектуальна транспортна система (ІТС) — це транспортна система, що використовує сучасні розробки в моделюванні та регулюванні транспортних потоків, що надає учасникам руху більшу інформативність та безпеку. ІТС якісно підвищує рівень взаємодії учасників руху в порівнянні зі звичайними транспортними системами.

Сукупність ІТС — це системна інтеграція сучасних комунікаційних інформаційних, технологій та засобів автоматизації з транспортною інфраструктурою, транспортними засобами та учасниками, яка орієнтована на підвищення безпеки й ефективності транспортного процесу, комфортності для водіїв та користувачів транспорту.

Порівняно зі звичайною транспортною системою ІТС відрізняються автомобільною навігацією, сигналами дорожнього руху, системами управління, змінні знаки повідомлення, автоматичне розпізнавання номерних знаків, камерами швидкості для моніторингу додатків, таких як системи відеоспостереження безпеки, і до більш вдосконалених додатків, які інтегрують живі дані та зворотній зв'язок з ряду інших джерел, таких як системи наведення паркування, інформація про погоду, інформаційні системи, системи розморожування чи дезінфікування мостів тощо. Додатково розробляються методики прогнозування, що дозволяють вдосконалити моделювання та порівняння з історичними базовими даними.

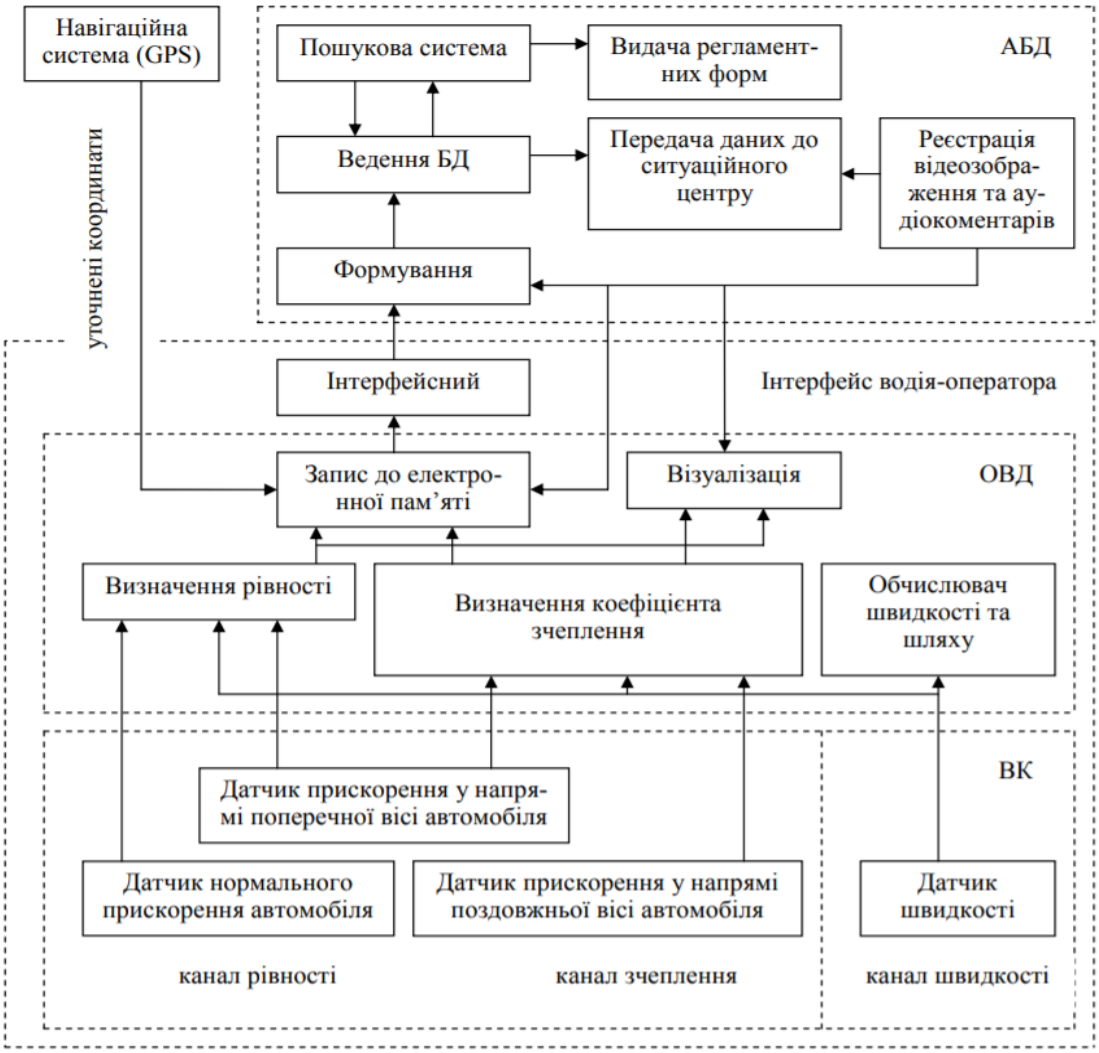


Рисунок 1.2 — Функціональна схема внутрішньої автомобільної телематичної системи

Сьогодні сфера просування ІТС у світовій практиці варіюється від вирішення проблем загального транспорту, істотного підвищення безпеки дорожнього руху, ліквідації заторів у транспортних мережах, підвищення продуктивності транспортної системи до екологічних та енергетичних проблем. Розвиток ІТС згруповано по шести пріоритетних напрямів:

- Оптимального використання інформації про дорогах, рух і поїздках, яке передбачає отримання актуальної і перевіреної інформації на всіх рівнях управління транспортом незалежно від форми власності оператора і виду транспорту і забезпечує її доступність для всіх користувачів;

- Забезпечення умов для без бар'єрного руху товарів і оптимального управління вантажними перевезеннями на європейських транспортних коридорах і в міських агломераціях за рахунок автоматичної ідентифікації транспортних одиниць, надання різних послуг (наприклад, митних) в режимі онлайн і просторового позиціонування на основі космічних навігаційних систем;
- Підвищення безпеки дорожнього руху за рахунок розвитку автоматичних систем, що попереджають і запобігають небезпечні ситуації як між транспортними засобами, так і між автомобілями і пішоходами;
- Забезпечення безпеки і захисту даних, що передаються в ІТС, зокрема особистих і фінансових даних користувачів;
- Інтеграції транспортного засобу в транспортну інфраструктуру за рахунок використання відкритих додатків в комп'ютерних системах транспортних засобів та програмному забезпеченні ІТС, що дозволяє забезпечити сумісність інформаційних систем і автоматично передавати дані, необхідні для оптимального управління як індивідуальними транспортними засобами, так і їх потоками.

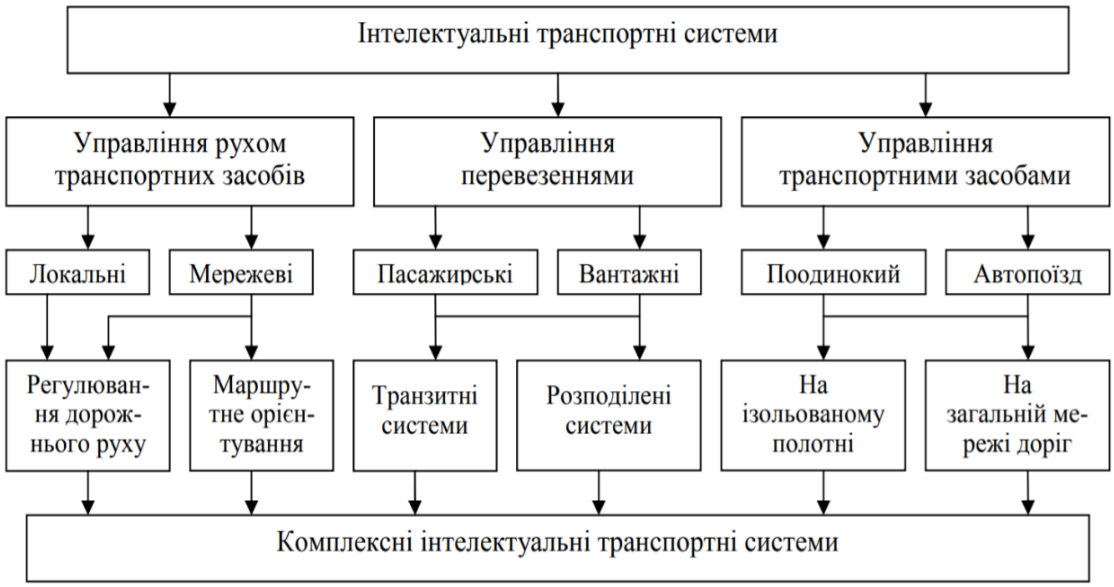


Рисунок 1.3 — Напрямки автоматизації транспортних систем

У сучасних програмах ІТС реалізується функція з передачі інформації і здійснення моніторингу з ряду технічних параметрів транспортних засобів, як щодо їх бортових датчиків, так і щодо бортових комп'ютерів - контролерів електронних систем керування робочими процесами вузлів, агрегатів та систем автомобіля. При цьому основними технічними складовими виступають засоби телематики, орієнтовані на отримання і передачу інформації з метою вирішення завдань, пов'язаних з організацією дистанційного діагностування технічного стану транспортних засобів. ІТС були створені для вирішення наступних задач:

- Оптимізації алгоритмів керування світлофорними об’єктами;
- Автоматичної фіксації порушень правил дорожнього руху;
- Надання пріоритету руху громадського транспорту;
- Моніторингу умов руху в режимі реального часу;
- Інформування учасників руху про дорожні умови;
- Інформування щодо наявності вільних паркувальних місць;
- Управління рухом громадського транспорту;
- Інформування щодо графіків руху громадського транспорту;
- Прогнозування транспортної ситуації на основі зібраних даних;
- Формування оптимальних маршрутів переміщення для учасників транспортного руху. [3]

Але для вирішення даних задач ІТС потребує даних з детекторів руху транспорту в режимі реального часу, з камер відео нагляду в режимі реального часу, GPS-датчиків служб таксі та громадського транспорту, швидкості руху транспортних засобів, оперативних даних щодо ДТП, ремонтів та перекриття вулиць і доріг. Система обміну даними представлена на рисунку 1.4.

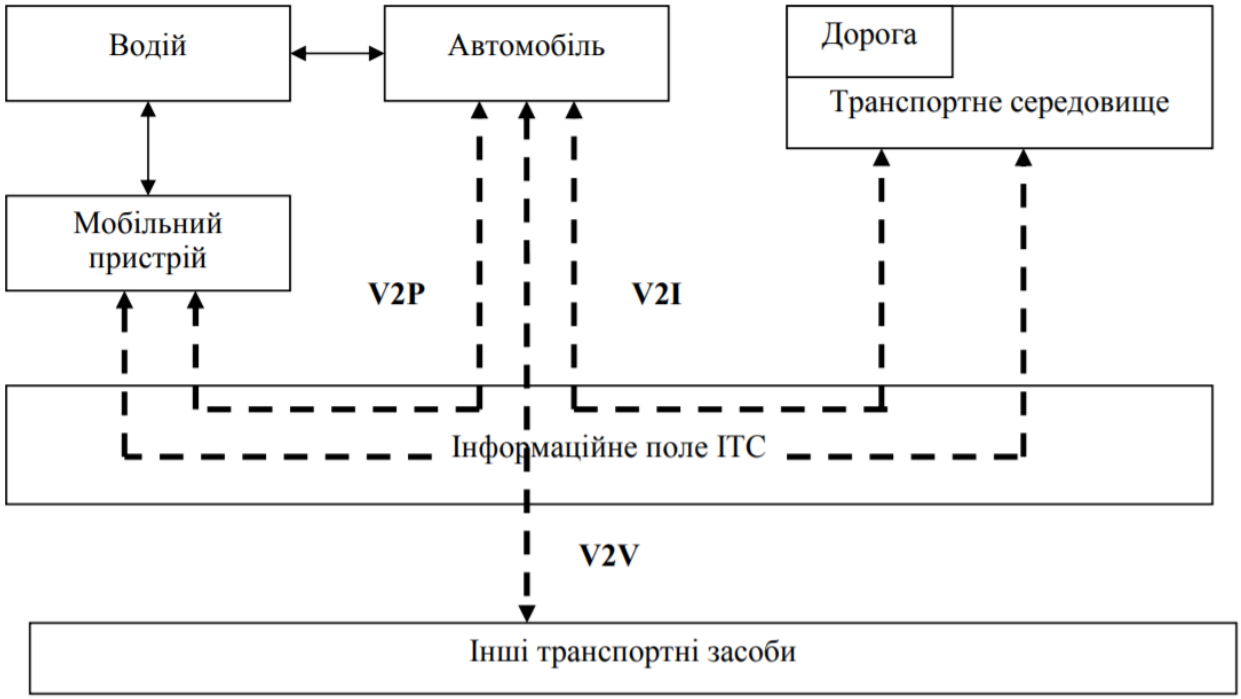


Рисунок 1.4 — Схема обміну інформаційними даними

Багато запропонованих систем ITC передбачають також нагляд за дорожніми шляхами, що є пріоритетом внутрішньої безпеки. Фінансування багатьох систем надходить або безпосередньо через організації національної безпеки, або за їх затвердженням. Крім того, ITC може відігравати певну роль у швидкій масовій евакуації людей у міських центрах після великих подій унаслідок нещасних випадків, таких як наслідки стихійного лиха чи загрози. Значна частина інфраструктури та планування, пов'язаних із ITC, є паралельною потребою в системах вітчизняної безпеки. [4]

Найбільш наочно можливості ITC представлені в системах PRT (Personal Rapid Transit), PAT (Personal Automated Transport). Це системи громадського транспорту, які забезпечують безупинну перевезення пасажирів на їх запит за допомогою автоматичних транспортних засобів без водія. Система PRT використовує власну транспортну мережу, яка може бути виконана у вигляді дорожнього полотна з напрямними пристроями, рейкового шляху або монорейки, а також у вигляді комбінації цих пристроїв. Особливістю системи є те, що вона перевозить пасажирів від початкової до кінцевої точки без

зупинки. Тобто користувач або група людей на зупиночному пункті обирає пункт призначення, і система подає вільний вагон відповідно до обраного маршруту. Також вільні вагони очікують пасажирів на станціях. Вагон з урахуванням топології мережі самостійно вибирає найкоротший шлях до пункту призначення. Вся система має централізоване комп'ютерне управління на рівні розподілу вагонів і забезпечення безпеки.

Перша система PRT експлуатується з 1975 р в місті Моргані в США, де пов'язує навчальні будівлі місцевого університету з декількома комплексами студентських гуртожитків. Загальна протяжність мережі 13,9 км, на якій є сім зупиночних пунктів. В системі експлуатується 73 повністю автоматичних вагонів. Вагони системи вміщають 20 чоловік і пересуваються по бетонному полотну з направляючими зі швидкістю до 30 км / ч. Вартість системи становила понад 60 млн доларів США. Система безкоштовно обслуговує 20 тис. студентів, а для жителів міста разова поїздка коштує 50 центів. З огляду на те що система проектувалася на початку 1970-х рр., Вона не має повного централізованого комп'ютерного управління, що компенсується роботою трьох диспетчерів. [5]

Інтелектуальна транспортна система використовує технології для покращення потоку руху. Розумна мережа взаємопов'язаних датчиків та розумних пристроїв може прокласти шлях до більш стійкого, ефективного та надійного центру міста. Вона створює ґрунт для майбутнього міста, до якого можна швидко проїхати, використовуючи поєднання традиційних та нових варіантів громадського транспорту. Системи ITC при повному впровадженні та широкому використанні споживачів можуть оптимізувати транспортний потік до точки, коли приватні автомобілі не знадобляться, а потреба в паркувальних місцях та багаторічних дорогах буде різко зменшена. Дороги, повні автомобілів, випарів газу та заторів, можуть бути минулим.

ВИСНОВКИ ДО РОЗДІЛУ 1

Отже, транспортна система – це система взаємопов’язаних складових (людей, які задіяні в транспортному процесі, інфраструктури, транспортних засобів тощо), яка призначена для транспортування будь-кого (будь-чого), від якої залежить більшість аспектів нашого життя. Будь то особисті мотиви чи потреби держави, незалежно від поставленої задачі контроль навантаження транспортної системи займає ключове місце.

Інтелектуальна транспортна система – це транспортна система з автоматизованими процесами, які створені для покращення взаємодії учасників та її оптимізації для досягнення максимальної ефективності використання. Якість покращення такої системи залежить від апаратної складової (кількості встановлених датчиків, камер чи будь-яких електронних пристроїв для збору транспортної інформації) та програмного забезпечення (GPS, балансування трафіку, системи інформування про ДТП, закриття дорожніх вузлів тощо).

РОЗДІЛ 2
АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ

2.1 Огляд існуючої проблеми

Сучасний світ важко уявити без автотранспорту. За результатами різних досліджень людина за своє життя проводить 3.5 – 4 років в дорозі. А скільки сфер впливу було розглянуто в першому розділі, які залежать від коректної роботи транспортної системи, починаючи з власних мотивів закінчуючи економічною складовою міст, держав та світу в цілому. Поміж всіх факторів які впливають на її роботу одним з основних факторів ефективної роботи транспортної системи є розподілення трафіку по системі. Але порівнюючи інтернет трафік та трафік наземного транспорту можна виділити деякі особливості при балансуванні:

1. Неможливість чіткого вибору шляху для подолання. Так як в інтернет мережах ми можемо встановити чіткі правила для розподілу і контролювати та направляти запит в залежності від завантаженості системи серверів, водія ми не можемо змусити їхати певною дорогою, якщо це казати в межах неавтоматичної некерованої складової системи. Але якщо водій зацікавлений приїхати якомога швидше і наш алгоритм дає такі можливості, то логічним було б його використання.

2. Складність реалізації та її вартість. Завантаженість інтернет трафіка можливо вирішити доданням додаткової апаратної частини та оптимальним розподілом, яке порівняно дешевше та швидше ніж створення нових транспортних вузлів, додання безлічі датчиків, які збирають дорожні дані (кількість машин, наявність аварій, тощо), використання супутників та створення програмної системи для зберігання та опрацювання отриманих даних. Така модернізація потребує багато часу та коштів.

Враховуючи ціну помилки при створенні транспортних шляхів та нових можливостей системи та її учасників, було прийнято рішення створити

програму, яка моделює транспортну систему та завдяки алгоритму балансує трафік для заданих параметрів навантаження.

Так як транспортна система схожа за своєї сутті схожа на граф, тому можна представити її у вигляді графу де вершини графа будуть точки можливих дорожніх розгалужень (поворотів, перехресть, тощо), а ребра транспортними шляхами.

Граф — це сукупність об'єктів із зв'язками між ними. Об'єкти розглядаються як вершини, або вузли графу, а зв'язки — як дуги, або ребра. Для різних областей використання види графів можуть відрізнятися орієнтованістю, обмеженнями на кількість зв’язків і додатковими даними про вершини або ребра. Враховуючи особливість транспортної системи ми не можемо використовувати звичайний граф, так як нам потрібно враховувати всі можливі перетини, тому за основу даної системи було взято планарний граф.

Планарний граф — це граф, який може бути зображений на площині без перетину ребер. Граф зображений на площині називається плоским, якщо його ребра не перетинаються. Граф називається планарним, якщо він ізоморфний деякому плоскому графу. Тобто існує відображення вершин графа на деякі точки площини і ребер графа на прості криві у площині, так що кінцями кривих є точки, що відповідають вершинам ребра і дві різні криві не мають спільних точок, окрім можливо кінцевих. [6]

Так як в основі задачі балансування трафіку лежить задача пошуку найкращого, оптимального шляху на основі системних даних про дистанцію та завантаженість транспортних вузлів між початковою та кінцевою точками маршруту, потрібно розглянути існуючі алгоритми пошуку оптимального шляху.

2.2 Алгоритми пошуку оптимального шляху

2.2.1 Алгоритм Дейкстри

Алгоритм Дейкстри (або алгоритм Найкоротшого першого шляху, алгоритм SPF Dijkstra) - це алгоритм пошуку найкоротших шляхів між вузлами в графі. Його винайшов комп'ютерний вчений Едсгер У. Дейкстра в 1956 р. і опублікував його через три роки. Алгоритм існує у багатьох варіантах. Оригінальний алгоритм Дейкстри знайшов найкоротший шлях між двома заданими вузлами, але більш поширений варіант фіксує один вузол як "джерельний" вузол і знаходить найкоротші шляхи від джерела до всіх інших вузлів у графі, створюючи найкоротший шлях дерево. Алгоритм описується наступною логікою:

1. Позначити всі вузли як невидимі. Створити список усіх нерозвіданих вузлів, які називаються невідвіданими множинами.
2. Призначити кожному вузлу орієнтовне значення відстані: нуль для нашого початкового вузла та на нескінченність для всіх інших вузлів. Встановлюємо початковий вузол як поточний (рис 2.1).

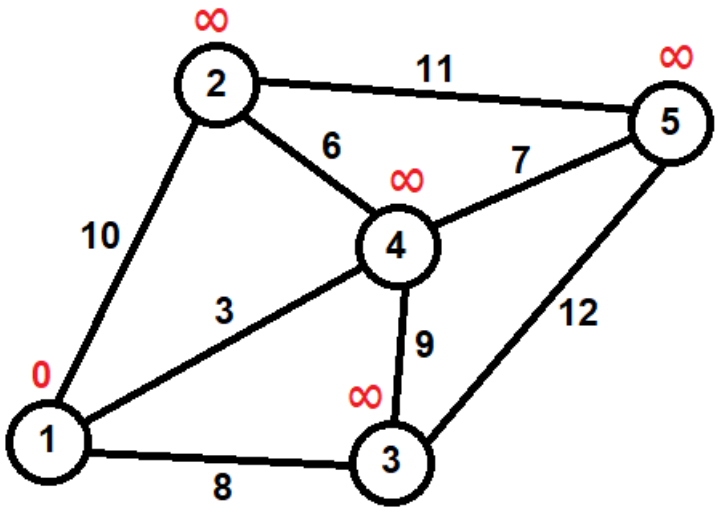


Рисунок 2.1 – Ініціалізація графу

3. Для поточного вузла розглянути всі його суміжні вершини та обчислити їх орієнтовні відстані через поточний вузол. Порівняти щойно

обчислену орієнтовну відстань із поточним призначеним значенням та призначити менший (рис 2.2).

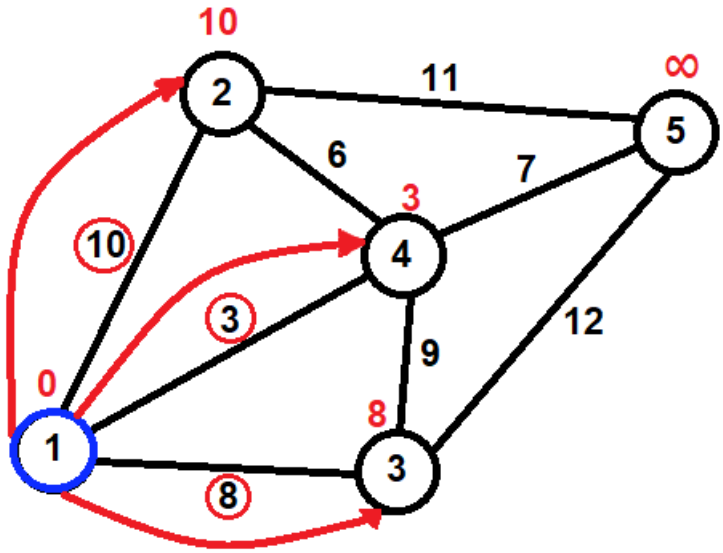


Рисунок 2.2 – Розгляд всіх суміжних вершин зі знаходженням відстані до них

4. Коли ми розглянули всі невідвідані суміжні вершини поточного вузла, позначаємо поточний вузол як відвіданий та видаляємо його з невідвіданого набору. Відвіданий вузол більше ніколи не перевірятиметься (рис 2.3).

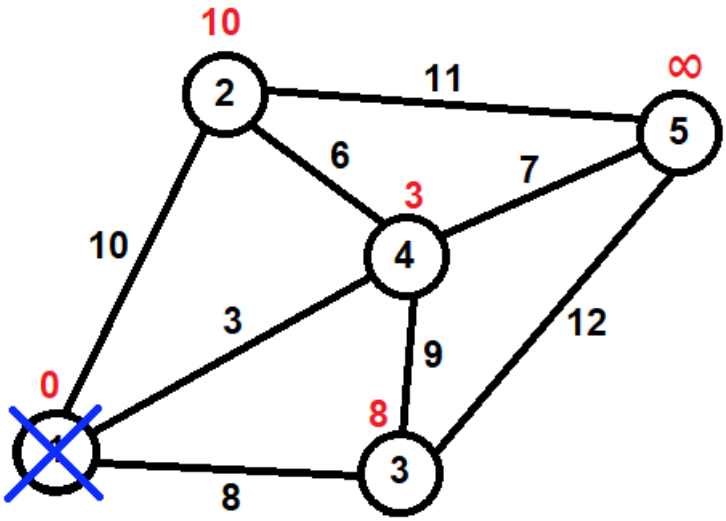


Рисунок 2.3 – Видалення відвіданої вершини

5. Якщо вузол призначення був позначений відвідуваним (при плануванні маршруту між двома конкретними вузлами) або якщо найменша орієнтовна відстань серед вузлів у невідомій множині - нескінченність (при

плануванні повного обходу; відбувається, коли між початковим вузлом немає зв’язку і залишилися невидимими вузлами), алгоритм вважається закінченим.

6. В іншому випадку обираємо невідвіданий вузол, який позначений найменшою попередньою відстані, встановлюємо його як новий "поточний вузол" та повертаємося до кроку 3.

Складність алгоритму Дейкстри залежить від способів накладення вершин v, а також можливість зберігання множинних неподільних вершин та способів оновлення міток. Позначимо n - кількість вершин, а m - кількість ребер в графі G. У випадку, коли для пошуку вершин з мінімальною вагою переглядається велика кількість вершин, а для зберігання величин ваг використовується масив, час роботи алгоритму є $O(n^2)$. Основний цикл виконує n операцій, при цьому в кожному з них при знаходженні мінімуму, виконується n операцій. Цикли кожної суміжної вершини, тратиться кількість операцій, пропорціональна кількості ребер m. Таким чином, загальний час роботи алгоритму дорівнює $O(n^2 – m)$, но враховуючи що $m < n * (n – 1)$ отримуємо $O(n^2)$. [7] Результати роботи алгоритму наведені на рис. 2.4.

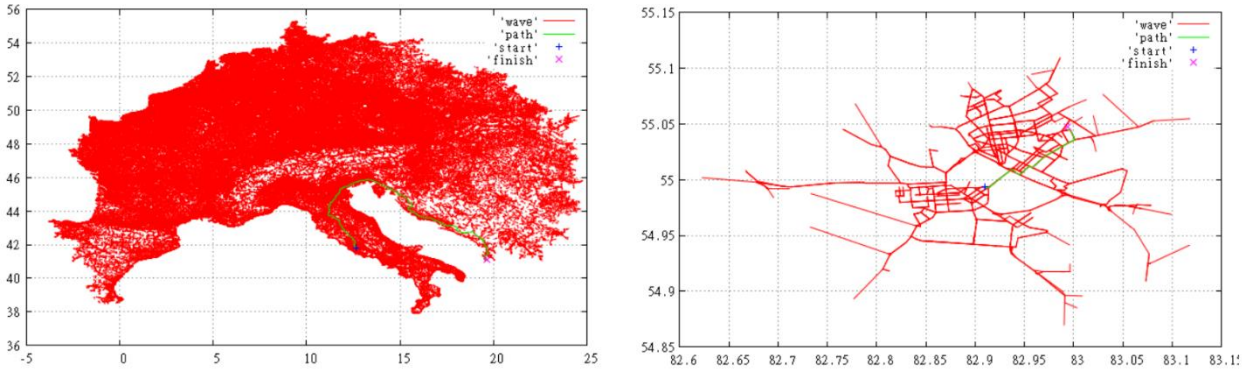


Рисунок 2.4 – Результати пошуку шляху з використанням алгоритму Дейкстри

Аналізуючи рис. 2.1.4 можна помітити що проблема алгоритму в тому, що алгоритм Дейкстри не використовує ніякої інформації про властивості графа і шуканого маршруту і при його роботі (поширення так званої хвилі) периметр цієї хвилі рухається від шуканої точки у всіх напрямках без будь-

якої дискримінації, що збільшує навантаження на пам’ять та швидкість пошуку оптимального маршруту між заданими точками.

2.2.2 Двонаправлений пошук

Двонаправлений пошук - це алгоритм пошуку графіків, який знаходить найкоротший шлях від початкової вершини до вершин цілі в спрямованому графіку. Він здійснює два одночасні пошуки: один від початкового стану і один від кінцевої вершини, зупиняючись, коли вони зустрічаються. Причина такого підходу полягає в тому, що в багатьох випадках він швидший: наприклад, у спрощеній моделі складності проблеми пошуку, коли обидва пошуки розширюють дерево з коефіцієнтом розгалуження b, а відстань від початку до цілі d, кожен із два пошуки мають складність O ($n^{d/2}$), і сума цих двох пошукових разів набагато менша, ніж складність O (n^d), яка виникла б від одного пошуку від початку до мети. Алгоритм складається з :

- прямого пошуку, аналогічного одиночного пошуку;
- зворотного пошуку;
- операції визначення належності списку іншому дереву пошуку.

Як і в пошуку A*, двонаправлений пошук може керуватися евристичною оцінкою відстані до мети (у передньому дереві) або від початку (у відсталому дереві). Результати роботи пошуку оптимального маршруту між містами за допомогою двонаправленого пошуку наведено на рис 2.5.

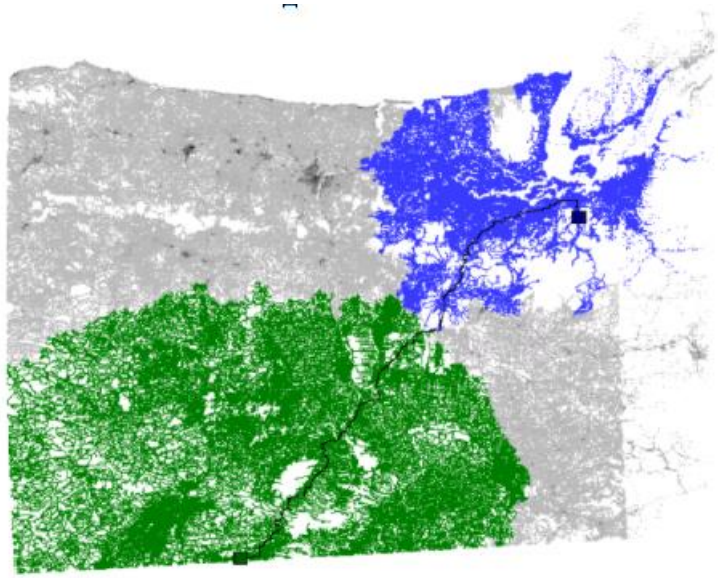


Рисунок 2.5 – Результат роботи двонаправленого пошуку на основі алгоритму A*

Справді, при гарній зв'язності хвиля являє собою еліпсоїд, дві малі хвилі, пущені назустріч, заметуть меншу площу в порівнянні з одним великим. З іншого боку, виникає проблема виявлення зустрічі хвиль, точок в їх периметрах може бути досить багато і перевіряти на кожному кроці наявність ребра в чужому периметрі не так вже й дешево. Можливо, з цим можна було б і змиритися при реальному виграші в обсязі переглянутої частини графа. Але от якщо ми розглянемо вищеописаний приклад пошуку (рис. 2.2.1) проїзду, то виявимо, що двонаправлений пошук нам нічим не допоможе, а тільки погіршить ситуацію, так як ми розглянемо область біля початку зустрічних шляхів, перш ніж хвилі зустрінуться. Бо замість однієї «хвилі», упершись в перешкоду, матимемо дві таких. [8]

2.2.3 A* алгоритм пошуку шляху

A* - це алгоритм проходження графів та пошуку шляхів, який часто використовується в інформатиці завдяки своїй повноті, оптимальності та ефективності. Одним з головних практичних недоліків є його потреба в пам’яті, оскільки вона зберігає всі створені вузли в пам'яті. Таким чином, у

практичних системах маршрутизації руху, як правило, перевершують алгоритми, які можуть попередньо обробити графік для досягнення кращої продуктивності, а також обмеженості пам'яті. Проте A* все ще є найкращим рішенням у багатьох випадках. Вперше алгоритм був опублікований Пітером Хартом, Нілсем Нілссоном та Бертрамом Рафаелем із Стенфордського науково-дослідного інституту у 1968 році. Це можна розглядати як розширення алгоритму Едсгера Дейкстри. A* досягає кращих показників, використовуючи евристику для керування його пошуком.

A* - алгоритм інформованого пошуку, що означає, що він формується у вигляді зважених графіків: починаючи з конкретного початкового вузла графіка, він спрямований на пошук шляху до заданого вузла цілі, який має найменшу вартість (найменша проїхана відстань, найкоротший час тощо). Це робиться, підтримуючи дерево шляхів, що виникають у початковому вузлі, і розширюючи ці контури один за одним він обходить граф поки не знайде задану вершину. При кожній ітерації свого основного циклу A* потрібно визначити, який з його шляхів слід продовжити. Це робиться на основі вартості шляху та оцінки вартості, необхідної для розширення шляху до цілі. Зокрема, A* вибирає шлях, який мінімізується. [9]

Типові реалізації A * використовують пріоритету чергу для виконання повторного вибору вузлів мінімальної (розрахункової) вартості для розширення. Ця чергу пріоритетів відома як відкритий набір. Алгоритм спершу відвідує ті вершини, які ймовірно ведуть до найкоротшого шляху до мети. Аби розпізнати такі вершини, кожній відомій вершині n співставляється значення F(n), яке дорівнює довжині найкоротшого шляху від початкової вершини до кінцевої, який пролягає через обрану вершину. Вершини з мінімальним значенням f обираються першочергово. Функція F(n) визначається як сума функція вартості від початкової вершини до n (g(n)) та евристичної функції (h(n)), яка оцінює вартість шляху від вершини n до кінцевої.

Існує два види евристичних функцій, які використовують в алгоритмі A*: припустимі та монотонні. Монотонні евристики також припустимі. Монотонність є сильнішою характеристикою припустимості. Зазвичай, використовують монотонні евристичні функції. Наприклад, пряма відстань між двома містами монотонна.

Евристика називається припустимою, якщо вона не переоцінює вартість маршруту. Тобто, оцінка шляху має знаходитись в проміжку фактичній вартості. Якщо використана евристика припустима, але не монотонна, тоді для досліджених вершин може бути невідомий найкоротший шлях. Тому має зберігатись можливість повторно досліджувати таку вершину.

Монотонна евристика має відповідати двом умовам: не переоцінювати вартість (аби бути припустимою) для кожної вершини n та суміжної до неї вершини n має виконуватись нерівність h(k) <= c(k, k`) + h(k`), де c(k, k`) фактична відстань між k та k`.

Тому ефективність даного алгоритму здебільшого залежить від обраної евристичної функції, в разі невдалого вибору, алгоритму потрібно буде обійти більше вершин для отримання результату.

Реалізація алгоритму A * включає підтримку двох списків – відкритий (open) та закритий (close). Відкритий містить ті вузли, які були оцінені евристичною функцією, але ще не розширені в наступники. Закритий містить ті вузли, які вже відвідали.

1. Визначаємо open список. Спочатку він складається з одної, початкової вершини.

2. Видаляємо вершину n з найменшим значенням f (n) з open списку і переміщуємо до close. Якщо вершина n – наша ціль, повертаємо результат.

3. Розгортаємо вузол n. Якщо будь-яка суміжна вершина до n – шукана вершина, повертаємо результат, простеживши шлях.

4. Для кожної наступної вершини застосовуємо функцію оцінювання. Якщо вершини не було в жодному списку, додаємо її до open списку.

5. Повертаємось до 2 кроку.

Зазвичай алгоритм A* переглядає лише частину вершин. Однак, в лабіринтах швидкодія наближається до найгіршого випадку. Результати роботи алгоритму A* на прикладі пошуку шляху від одного міста до іншого наведені на рис. 2.6.

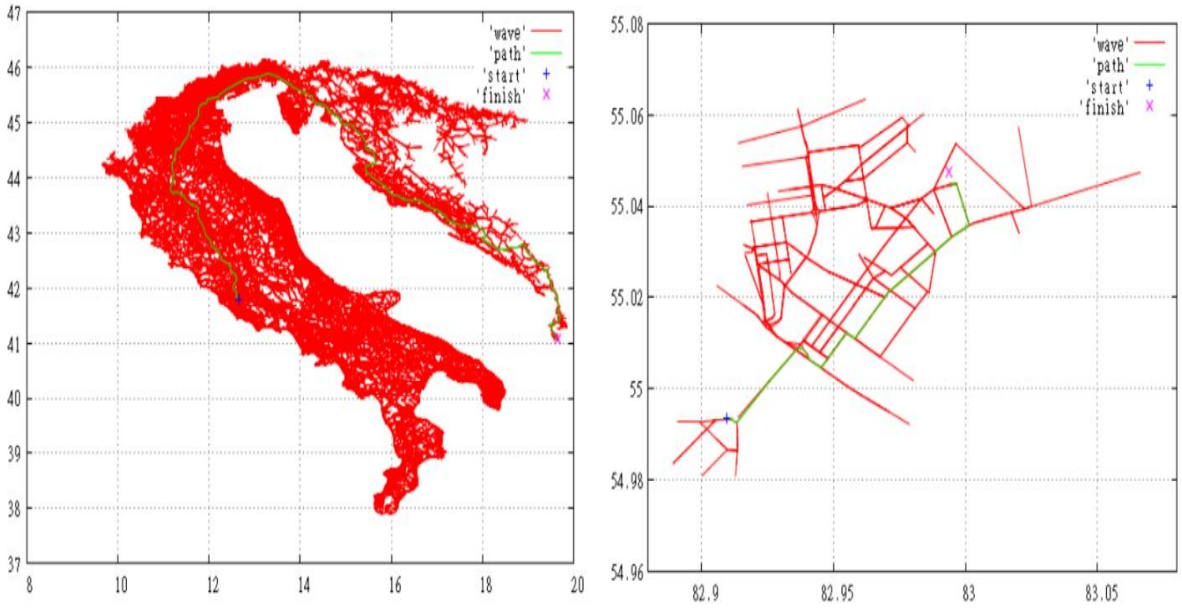


Рисунок 2.6 – Результат пошуку шляху алгоритмом A*

Порівнюючи результати наведені на рис. 2.6, та попередньо розглянуті результати алгоритму Дейкстри та алгоритму двонаправленого пошуку, можна помітити різницю в їх результатах, а саме в кількості опрацьованих шляхів для пошуку оптимального. Швидкодія алгоритму суттєво залежить від двох факторів:

- Точність евристичної функції (якщо евристика не монотонна, час роботи алгоритму буде експоненційним, оскільки вершини можуть переглядатись кількаразово тому, чим точніша оцінка вартості, тим менше вершин буде досліджено).
- Реалізація списків відомих та досліджених вершин (найбільш витратними операціями в алгоритмі є операції додавання, вилучення та зміни елементів в списках відомих та досліджених вершин, тому на їхню швидкодію істотно впливають конкретні реалізації цих структур даних).

Нехай G — множина вершин в графі, інформація про вершини та ребра доступна до початка роботи алгоритму; використана евристична функція — монотонна. Список відомих вершин реалізований як бінарна купа, список досліджених — як масив. Тоді алгоритм A* має квадратичний час роботи в найгіршому випадку $O(n^2)$.

Основним недоліком алгоритму A* є потреба в пам'яті для збереження всіх відомих та досліджених вершин. Через це алгортим A* непридатний для багатьох задач. Але існують різні модифікації стандартного алгоритму A*:

- IDA* – A* з ітеративним пошуком в глибину.
- RBFS – рекурсивний пошук за найкращим збігом, вимагає лінійну кількість пам'яті в залежності від довжини розв'язку.
- MA* – A* з обмеженням пам'яті, або SMA* з використовуванням лише наперед виділеним обсягом пам'яті.

Порівнюючи IDA* та RBFS, можна оцінити їх ефективність як еквіваленту, хоч RBFS є трохи ефективнішим, але при цьому він зберігає декілька більше інформації про шуканий шлях, що може відобразитись на великих системах.

Використання модифікацій алгоритму з обмеженням пам’яті (MA*, SMA*) є не зручним, тому що потрібно постійно рахувати розмір виділеної пам’яті для роботи алгоритму, що може призвести до нестабільної роботи системи.

Тому з вище перерахованих модифікацій алгоритму A* я пропоную розглянути IDA*, так як дана модифікація дозволяє значно зменшити використання пам’яті без надання особливих границь використання пам’яті.

2.2.4 IDA* алгоритм пошуку шляху

Ітераційне поглиблення A* (IDA*) – це алгоритм проходження графів і алгоритм пошуку шляху, який може знайти найкоротший шлях між визначеним початковим вузлом та будь-яким членом набору цілей вузлів у зваженому графі. Це варіант ітеративного поглиблення пошуку в глибину,

який запозичує ідею використовувати евристичну функцію для оцінки вартості, що залишилася, щоб досягти цілі за допомогою алгоритму пошуку A*. Оскільки це алгоритм пошуку на глибині споживання, його використання пам'яті нижче, ніж у A *, але на відміну від звичайного ітеративного поглибленого пошуку, він концентрується на дослідженні найбільш перспективних вузлів і, таким чином, не йде на ту саму глибину скрізь у дереві пошуку.

На відміну від A*, IDA* не використовує динамічне програмування, тому часто закінчує досліджування одних і тих же вузлів багато разів. Також на відміну від A*, нам не потрібно зберігати набір орієнтовних вузлів, які ви збираєтесь відвідати, тому споживання нашої пам’яті присвячено лише локальним змінним рекурсивної функції, тому часто завершає дослідження одних і тих самих вузлів багато разів і має складність поліноміального простору, точніше, O (n^m), де n - максимальний коефіцієнт розгалуження, а m - максимальна глибина дерева. IDA* все ще має експоненціальну часову складність A*.

На закінчення, IDA* має кращий об'єм пам'яті, ніж A*. Так як і в A*, і на відміну від IDDFS, він концентрується на дослідженні найбільш перспективних вузлів, і, таким чином, не йде на ту саму глибину скрізь у дереві пошуку. IDA* часто закінчує дослідження одних і тих же вузлів багато разів (як IDDFS), але, асимптотично, і A*, і IDA* мають однакову експоненціальну часову складність, тобто O (n^m). Тобто IDA* при поглибленні хибного шляху припинить подальший перегляд вершин за умови перебільшення функції евристики, в даному випадку абсолютної відстані між точками. На рис 2.7 наведено порівняльний приклад використання пам’яті двох алгоритмів

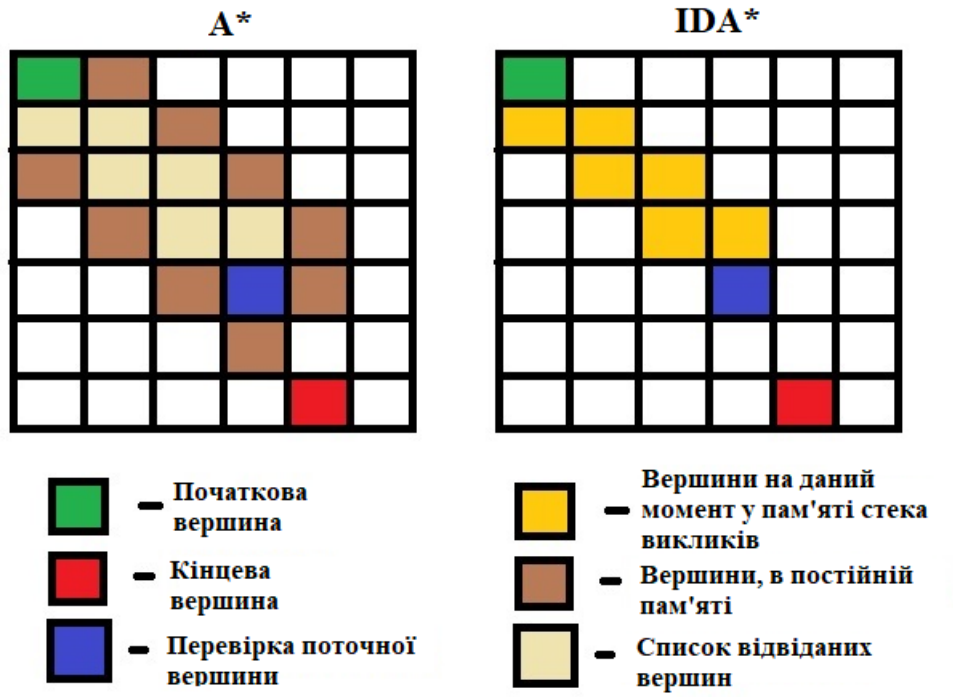


Рисунок 2.7 – Порівняння використання пам’яті алгоритмами A* та IDA*

2.3 Порівняльний аналіз розглянутих алгоритмів

Отже, який із них найкращий? Так як алгоритму Дейкстри потрібно переглянути всі вершини, він витрачає набагато більше часу, щоб знайти найкоротший шлях, а також відсутність евристики у використанні на великих системах може стати проблематичним, так як може з’явитися потреба в використанні великої кількості пам’яті. Двонаправлений пошук оптимальніший за алгоритм Дейкстри, так як не потрібно обходити весь граф, але навіть при використанні евристики, хвилі алгоритму можуть не зійтись при поганій зв’язності графу і буде програш по часу та пам’яті відносно однонаправлених алгоритмів пошуку. A* стає недоцільним, коли обсяг пошуку просто великий, через обмеження пам'яті. Тож у цих випадках IDA*, безумовно, більше підходить. Взагалі, IDA* - це одна з найкращих оптимальних методів пошуку простору стану навколо. Однак A* концептуально простіший, ніж IDA*. Як наслідок, на практиці A* може бути простішим у застосуванні, ніж IDA*, але, у реальному сценарії, це насправді не є проблемою, оскільки вони обидва порівняно легко реалізовуються, але так

як сучасні транспортні системи складаються з великої кількості вузлів, використання звичайного A* може спричинити виникнення проблем з використанням пам’яті, тому в своїй роботі я буду використовувати IDA*.

ВИСНОВКИ ДО РОЗДІЛУ 2

Було розглянуто опис існуючої проблеми та обрано метод її вирішення, а саме балансування трафіку в транспортній системі завдяки алгоритму пошуку оптимального шляху між точками, за допомогою якого буде відбуватися розподіл навантаження транспортної системи.

Проведено порівняльний аналіз існуючих алгоритмів пошуку оптимального шляху між двома точками. В ході нього виявлені плюси та мінуси кожного з них. Наведено приклади роботи кожного з алгоритмів, на основі яких було отримано рішення про використання алгоритму IDA*, так як він задовольняє умовам швидкодії та порівняно менше ніж інші використовує пам’ять. А так як сучасні транспортні системи представляють собою величезну кількість транспортних шляхів та вузлів, необхідно врахувати навантаження на систему балансування трафіку.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Створення концепції програмного продукту

Так як наша транспортна система представлена у вигляді планарного зваженого графу, для комфортного використовування було прийнято рішення про створення редактору, у якому можна було б швидко та зручно редагувати відкритий граф, та при необхідності зберегти в файл, для подальшого використання. Після потрібної конфігурації системи, програма повинна перейти до режиму балансування поточної системи, де, на основі введених користувацьких даних, буде детально описано та продемонстровано як саме відбувався процес балансування навантаження. Для створення програмного продукту я відокремив основні етапи розробки програми:

- Вибір інструментарію (мови програмування, бібліотеки, середовища розробки, тощо) та системи контролю версій програмного продукту.
- Створення відповідного функціоналу.
- Створення макету інтерфейсу програми.
- Тестування програми, виправлення помилок функціонування, розробка нових можливостей програми, після використання, для покращення зв’язку програма-користувач.

3.2 Вибір мови програмування

Проаналізувавши потреби поставленої задачі та вивчивши перелік популярних мов програмування, а саме C, C++, Java, було обрано мову програмування Python, так як вона дозволяє отримати бажані результати та ефективна в написанні коду програмного продукту. [10]

Python - інтерпретована мова програмування високого рівня, загального призначення. Створена Гідо ван Россумом та вперше випущена в 1991 році, філософія дизайну Python підкреслює читабельність коду завдяки помітному

використанню значного пробілу. Його мовні конструкції та об'єктно-орієнтований підхід мають на меті допомогти програмістам написати чіткий логічний код для малих та масштабних проектів. Він підтримує декілька парадигм програмування, включаючи структуроване (зокрема, процедурне), об'єктно-орієнтоване та функціональне програмування. Інтерпретатори Python доступні для багатьох операційних систем. [11] Серед плюсів даної мови програмування хотілось би виділити основні:

- Простота написання коду, та його так звана читабельність. [12] Непотрібно створювати безліч власних допоміжних структур для простих задач, як в багатьох інших мовах програмування, достатньо стандартного функціоналу для їх вирішення. Python пропонує велику стандартну бібліотеку, яка включає такі сфери, як Інтернет-протоколи, струнні операції, інструменти веб-служб та інтерфейси операційної системи. Багато завдань програмування з високим рівнем використання вже написані у стандартній бібліотеці, що значно скорочує довжину коду, який потрібно записати.
- Структури даних в Python мають вбудовані структури даних зі списку та словника, які можна використовувати для побудови швидких структур даних для виконання. Крім того, Python також надає можливість динамічного введення даних на високому рівні, що зменшує необхідну тривалість коду підтримки.
- Продуктивність та швидкість. Python має чистий об'єктно-орієнтований дизайн, забезпечує розширені можливості управління процесами, володіє сильними можливостями інтеграції та обробки тексту та власною рамкою тестування одиниць, що сприяє підвищенню швидкості та продуктивності. Python вважається життєздатним варіантом для створення складних багатопротокольних мережевих додатків. Крім того для прискорення роботи програми можлива інтеграція мови С в програмний код програми. [13]
- Безкоштовне використання. Мова Python розроблена під затвердженою OSI ліцензією з відкритим кодом, що дозволяє вільно використовувати та розповсюджувати, в тому числі для комерційних цілей.

- Доступна підтримка та розвиток. Широка база користувачів та активних розробників призвела до того, що багатий Інтернет-ресурсний банк сприяв розвитку та подальшому прийняттю мови. Завдяки розвитку якої було створено безліч корисних бібліотек на мові Python з відкритим кодом, а широке розповсюдження сприяє швидко знаходити відповіді на користувацькі запитання. Ще одним плюсом можна виділити документацію, яка в точності описує майже кожну функцію з її параметрами, що також пришвидшує процес розробки на даній мові.

Підводячи підсумки можна сказати що Python – зручна і потужна мова програмування. Вона має правильне поєднання продуктивності та можливостей, які роблять написання програм на Python одночасно простим та ефективним. А завдяки вбудованих бібліотек та безлічі інших, створених шанувальниками цієї мови, можна швидко і без проблем створити програму, яка б задовольняла поставлену мету.

Вибір середовища розробки (IDE) не є принциповим. Для Python типовим IDE є розроблене компанією JetBrains середовище під назвою Pycharm, яке користується популярності у Python розробників, але так як я звик використовувати Visual Studio Code, або VS Code, я буду використовувати саме її. Вона має безліч розширювань, в тому числі для обраної мною мови таких як перевірка написання правильності коду згідно прийнятими стандартами (PEP8 для Python), автодоповнення коду, тощо.

Для контролю версій своєї роботи було обрано GitLab, так як вона безкоштовна та зручна для використання та контролю версій програми.

3.3 Огляд бібліотек

Для створення графічного інтерфейсу програмного продукту я використав наступні бібліотеки мови програмування Python:

- Для створення GUI було використано вбудовану бібліотеку Tkinter. Це крос-платформна бібліотека, візуальні елементи відображаються за допомогою вбудованих елементів операційної системи, тому програми,

побудовані за допомогою Tkinter, виглядають так, як вони належать на платформі, де вони запущені. Легкий у використанні порівняно з іншими GUI бібліотеками. Має велику кількість графічних елементів та функціоналу, для створення GUI. [14]

- Так як стандартний функціонал бібліотеки Tkinter на жаль не має підтримки опису графічного елемента при наведенні курсора на нього, тому було прийнято рішення використання бібліотеки PMW. PMW - це інструментарій для створення складних графічних об’єктів високого рівня за допомогою модуля Tkinter. [15]
- Він складається з набору базових класів та бібліотеки гнучких та розширюваних великих графічних елементів, побудованих на цій основі.
- Для відображення транспортної системи під час процесу балансування було використано Matplotlib та Networkx . Бібліотека Matplotlib створена для візуалізації даних двовимірною або тривимірною графікою, є гнучким, легко настроюваним пакетом, який надає великі можливості, таких як побудова графіків, відображення графів, створення різного виду діаграм, гістограм, тощо. [16] Підтримує створення динамічного змінення графіку (анімація), запису в файл, в більшу частину форматів зображень. NetworkX є безкоштовною бібліотекою Python для дослідження графів та мереж. Має великий функціонал для роботи з графами. [17]
- Для генерації шляху анімації було використано бібліотеку NumPy. Один з основних додаткових пакетів наукових обчислень в Python. Він містить в собі потужний математичний арсенал (лінійна алгебра, перетворення Фур'є, випадкові числа, тощо) та підтримує вбудовані в неї великі багатомірні масиви та матриці. За рахунок того що вона використовує не тільки Python, а й мову програмування C, її обчислення набагато швидше ніж аналогічна реалізація на основі вбудованих структур та методів на Python. [18]

Поміж вище наведених бібліотек було використано декілька типових вбудованих Python бібліотек, таких як [19]:

- OS, для роботи з відкриттям, збереженням файлу транспортної системи, та файлового менеджеру користувацької системи.
- Json, для збереження та зчитування даних файлу.
- Collections, для використання структури defaultdict.
- Math, для підрахунку абсолютного шляху між точками, та іншими математичними операціями системи.
- Random, для додавання випадкової завантаженості в процесі балансування с керованим відсотком генерації.

3.4 Структура програми. Класи.

Програма складається з 3 класів Node, Graph та Interface, схема класів яких наведена в додатку 1.

Клас Node створений для зберігання суміжних до нього вершин з додатковою інформацією, а саме завантаженості ребра та дистанції між ними. Він має лише 1 функцію update_edge_weight, яка створена для оновлення даних вузла. Створюється для кожної вершини переданої до класу Graph.

Клас Graph створений для відображення процесу навантаження згідно переданими даним, які отримав клас Interface від користувача за допомогою networkx та matplotlib. Він створюється при ініціалізації класу Interface і являє собою поле graph. [20] Має такі основні методи:

- Ida_star – метод в якому описаний алгоритм IDA*, завдяки якому виконується балансування введеного навантаження на систему. Логіка якого базується на дистанції між вершинами та навантаженням їх вузлів. Евристичним алгоритмом було обрана абсолютна дистанція між точками. Тобто, наприклад дистанція між точкам 15 умовних одиниць, а коефіцієнт навантаження 2. При множенні дистанції на коефіцієнт завантаженості ми отримаємо результат в 30, тобто на подолання дистанції в 15 одиниць ми витратимо як на 30.

- Add_random_loading – даний метод працює, якщо обрано режим моделювання з додаванням довільного навантаження, тим самим демонструючи зміни по ходу навантаження вже введених даних.
- Animation – метод, який реалізовує візуалізацію балансування на основі методу Ida_star.
- Generate_animation_path – метод який генерує розмір анімаційного шляху в залежності від обраної користувачем швидкості відображення.
- Add_graph_loading – метод, який отримує передані користувацькі дані для подальшого балансування.
- Add_nodes – отримує топологію транспортної інтелектуальної системи від класу Interface для подальшого опрацювання.
- Update_loadding – оновлює інформацію про систему на графі в разі змін.
- Calculate_distances – метод, який вираховує абсолютну відстань між введеною вершиною до всіх інших.

Клас Interface найбільший, серед наведених, так як більшість функціоналу програми зв’язано з інтерфейсом. Більшість якого написано на вбудованій бібліотеці Tkinter з використанням різних графічних об’єктів, а саме Menu, Frame, Canvas, Button, Menubutton, PhotoImage, Label, Entry, Toplevel, messagebox, simpledialog, Scrollbar. Він має такі основні методи:

- Init_interface_objects – метод, який створює майже всі об’єкти користувацького інтерфейсу, та викликається лише 1 раз при виклику методу start(), щоб запобігти пере об’явленню.
- Mouse_move_b1 – метод, який відслідковує всі натиснення лівої кнопки миші користувача на активному полотні редактора. Він призначений для переміщення вершин системи.
- Mouse_double_click_b1 – метод, який відслідковує всі подвійні натиснення користувача на активному полотні редактора. Даний метод

призначений для виділення вершин чи ребер, для утворення зв’язку між ними, редагування завантаженості ребра чи їх видалення.

- Editor_delete_object – видаляє раніше обраний елемент з редактору, якщо це вершина видаляє також всі її зв’язки.
- Create_edge – створює зв’язок між двома послідовно вибраними за допомогою подвійного натиснення вершини.
- Create_node – за допомогою кнопки в інтерфейсі створює нову вершину присвоюючи їй найменше можливе значення id.
- Find_path_input_data – відкриває вікно поверх основного вікна, в якому пропонує ввести дані для завантаження системи.
- Open_statistics – відкриває вікно поверх основного вікна, в якому відображає детальну інформацію відносно останнього навантаження.
- Change_display_speed – змінює швидкість відображення процесу балансування. Має значення від 1 до 6.
- Open_file – метод для відкриття файлу в форматі json. При помилці формату чи вмісту програма відобразить віконце з помилкою.
- Editor_save_topology – зберігає транспортну систему в json файл у вигляді словника: вершина: зв’язки, координати.
- Open_editor / Simulation – відкриває вікно редактору та вікно розподілення балансування відповідно. При запуску програми стандартно відкривається вікно редактору.
- Edge_move – використовується при переміщенні вершин. Розраховує нові позиції для змінених шляхів.
- Get_distance / recalculate_distance – методи для отримання та розрахунку нової дистанції між вершинами, в разі їх зміщення.
- Clear_editor – видалення всіх об’єктів у вікні редактору.

Також були реалізовані додаткові методи для контролювання відображення масштабу системи, перевірки вхідних даних, оновлення змін на графічному інтерфейсі, тощо.

3.5 Приклад роботи алгоритму

Система працює в двох режимах – з додаванням випадкового навантаження, та без додавання. В першому випадку алгоритм працює один раз, так як немає потреби в перерахуванні. У разі додавання випадкового навантаження, система кожен раз перераховує маршрут учасника, коли той наближається до вузла, якщо він не кінцевий. Це пояснюється тим, що ситуація на дорогах постійно змінюється і маршрут, який був спочатку розрахований може бути не оптимальним. Для пошуку оптимального шляху використовується алгоритм IDA*. Так як алгоритм передбачає евристичну функцію, тому в якості неї було взято функцію розрахунку абсолютної дистанції між пошуковими вузлами. Вага шляху визначається як дистанція між двома вершинами помножена на коефіцієнт навантаження.

Розглянемо принцип роботи алгоритму на прикладі випадково згенерованої системи з 5 вершинами. Структура системи та її параметри (ваги) наведені на рис 3.1 та рис 3.2.

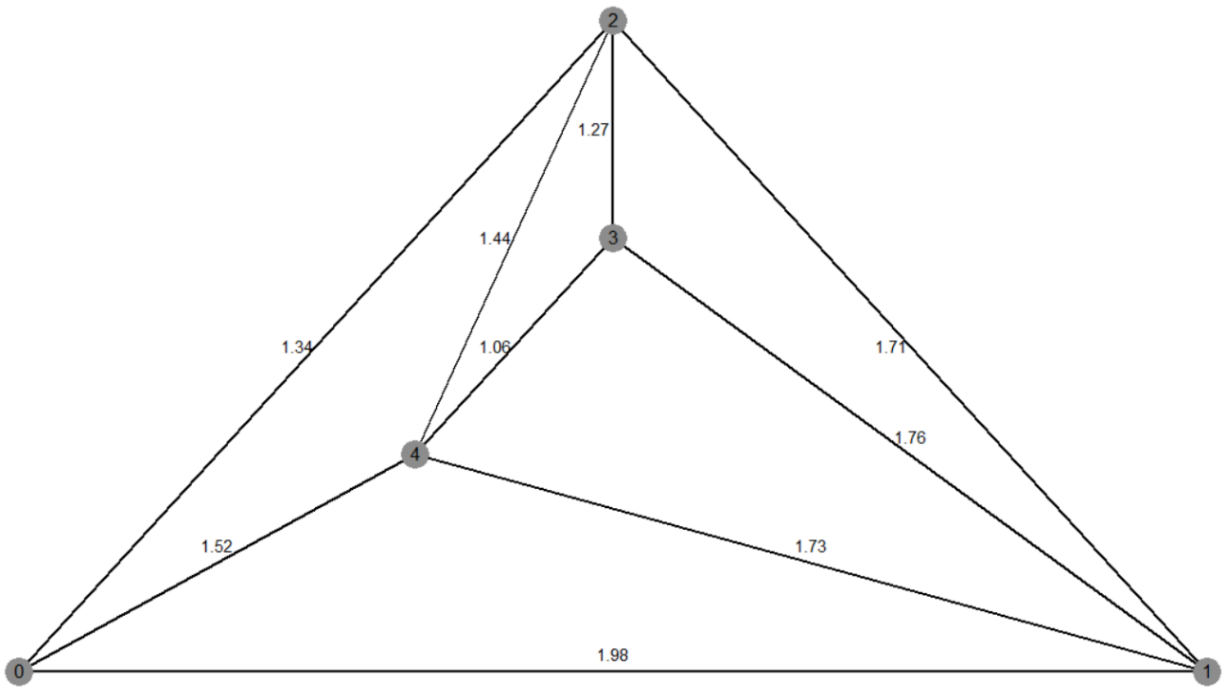


Рисунок 3.1 – Випадковий планарний граф з вагами навантаження

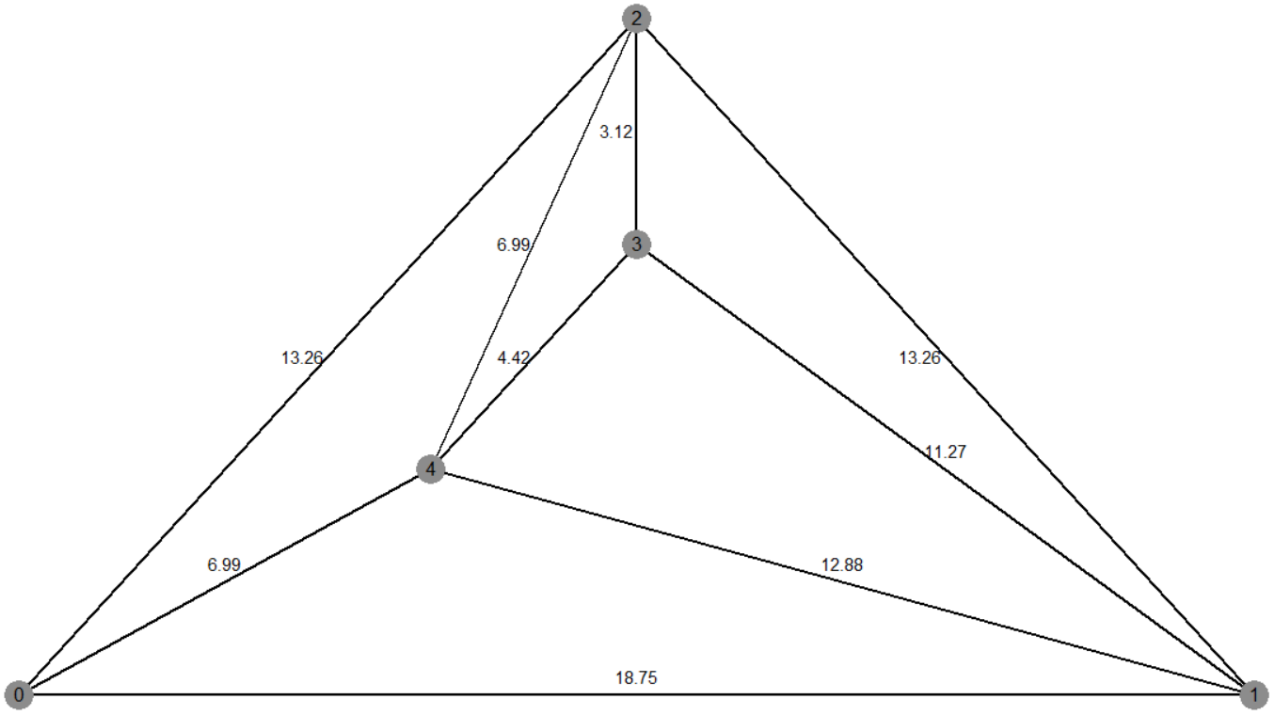


Рисунок 3.2 – Випадковий планарний граф з вагами дистанцій

Розглянемо покрокову роботу алгоритму без додання випадкової завантаженості на прикладі пошуку оптимального шляху від «0» до «3» вершини:

1. Створимо початковий порожній масив path, в якому будемо зберігати шуканий шлях та добавимо до нього початкову вершину «0». Початкову границю пошуку bound прирівнюємо абсолютній дистанції до вершини «3». Розраховуємо абсолютні дистанції від «3» вершини до всіх інших. Ці відстані будуть використовуватися для отримання абсолютної дистанції між поточними та шуканою вершинами. Результати продемонстровані на рисунку 3.4.

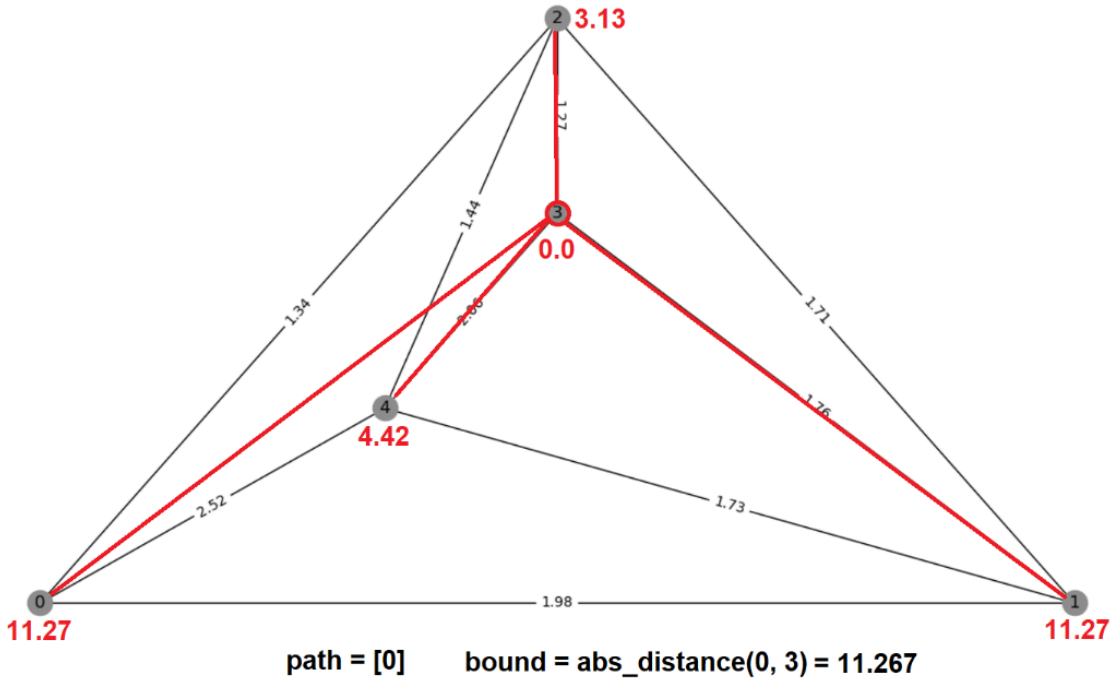


Рисунок 3.3 – Розрахунок дистанцій від кінцевої точки до всіх інших та задання стартових величин

2. Рахуємо оцінку мінімальної вартості шляху f за формулою f = g + distances[current_node], де g – поточна вартість шляху, distances[current_node] – абсолютна дистанція від шуканої вершини до поточної.

Порівнюємо з bound, якщо f більше заданого порогу, то повертаємо це значення і встановлюємо нову величину bound = f. Якщо ця вершина є шукана повертаємо результат path, якщо ні розглядаєм всі суміжні вершини.

3. Для кожної суміжної вершини рекурсивно виконуємо крок під номером 2, додаючи до поточної оцінки вартість шляху суміжної вершини помноженої на її завантаження (g = g + loading[current_edge] * distance[current_edge]), поки не знайдемо шукану вершину. Якщо bound перевищує шуканий ліміт повертаємо Not found.

Результати роботи алгоритму наведені нижче:

f: 11.267, current bound: 11.267, path: [0];

f: 48.392, current bound: 11.267, path: [0, 1];

f: 20.893, current bound: 11.267, path: [0, 2];

f: 15.044, current bound: 11.267, path: [0, 4];

path: [0], new bound: 15.044;

f: 11.267, current bound: 15.044, path: [0];

f: 48.392, current bound: 15.044, path: [0, 1];

f: 20.893, current bound: 15.044, path: [0, 2];

f: 15.044, current bound: 15.044, path: [0, 4];

f: 44.175, current bound: 15.044, path: [0, 4, 1];

f: 23.815, current bound: 15.044, path: [0, 4, 2];

f: 15.31, current bound: 15.044, path: [0, 4, 3];

path: [0], new bound: 15.31;

f: 11.267, current bound: 15.31, path: [0];

f: 48.392, current bound: 15.31, path: [0, 1];

f: 20.893, current bound: 15.31, path: [0, 2];

f: 15.044, current bound: 15.31, path: [0, 4];

f: 44.175, current bound: 15.31, path: [0, 4, 1];

f: 23.815, current bound: 15.31, path: [0, 4, 2];

f: 15.31, current bound: 15.31, path: [0, 4, 3];

result: [0, 4, 3].

3.6 Огляд створеного програмного продукту

Програма складається з 2 основних вікон – редактору та вікна симуляції. Схема алгоритму програми наведена в додатку 2. Вікно редактору містить поле для редагування системи та панель з інструментами, яка містить кнопку створення вершини, очищення редактору, симуляції, індикатори масштабування та вибір відображення обраних ваг системи. Поле редактору було створено на основі Canvas бібліотеки Tkinter. Решта графічних об’єктів являють собою кнопки, підписи та області (Button, Label, Frame). При натисненні кнопки “Створити вершину” з’являється вершина з мінімальним доступним номером вузла в лівому верхньому кутку екрана. Демонстрація приведена на рис 3.4.

Рисунок 3.4 – Головне вікно редактору зі створеною вершиною

При створенні нової вершини на новому полотні використовується масштаб 1 піксель дорівнює 2 одиниці дистанції, а при уже завантаженій системі, береться будь-яке ребро і вираховується масштаб відповідно відстані згідно координат, та введеному числу реальної дистанції між ними. При переміщенні вершин відбувається перерахування дистанції згідно масштабу рис. 3.5.

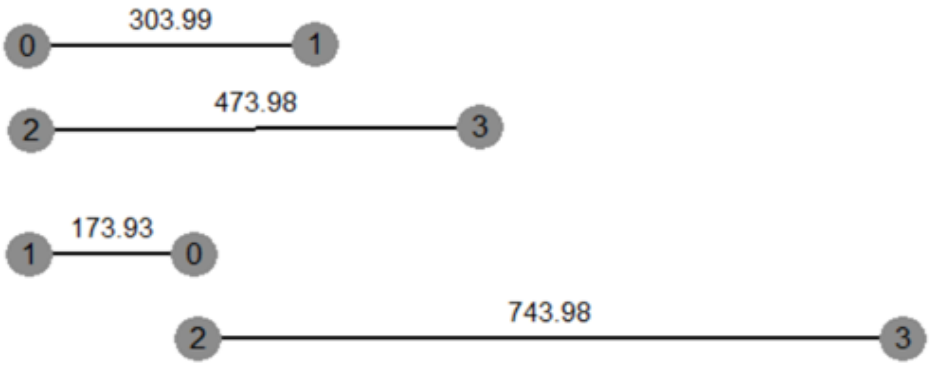


Рисунок 3.5 – Перерахування дистанції при переміщенні вершин

При подвійному натисканні на вершину в полі редактору вона додається до можливої пари з’єднання і зберігається у кеші програми, доки не буде створений зв’язок, і видаляється з пари при виконанні інших дій, тобто неможливо одночасно виділити і вершину і ребро. Вибір ребра працює лише для видалення його при натисненні клавіші delete. При виборі елемента він додається до кешу як зараз обрано, і фарбується в світліший колір для відображення обраних елементів, це продемонстровано на рис 3.6.

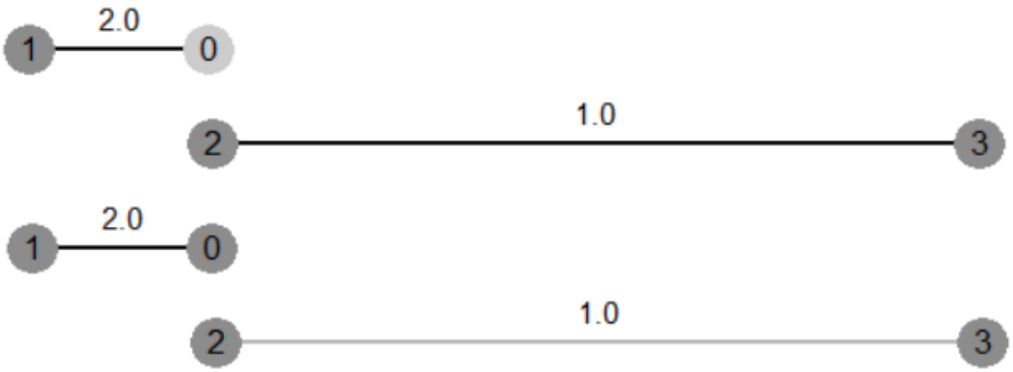


Рисунок 3.6 – Вибір елементу

Також подвійне натискання працює на підпис ребра, але лише для ваг навантаження. При цьому з’являється вікно для вводу нового значення для даного ребра, яке використовує функцію для перевірки користувацьких введених даних рис 3.7.

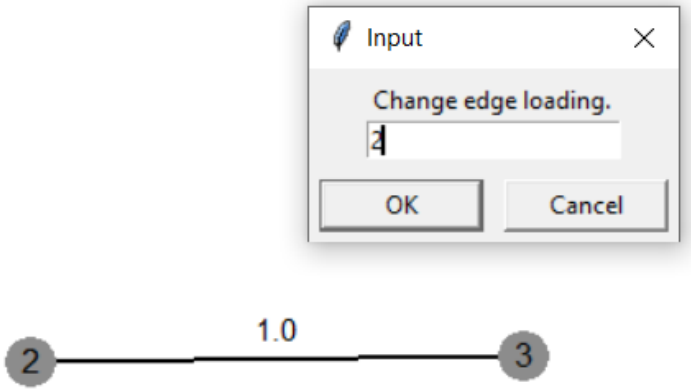


Рисунок 3.7 – Зміна значення навантаження для ребра.

Так як інтелектуальні транспортні системи можуть досягати великих розмірів, було прийнято рішення інтегрувати масштабування, для зручної та детальної конфігурації системи, рис 3.8 та рис 3.9. Серед особливостей масштабування хотілось би виділити розмір вершин, які при масштабі більше 100% не змінюють своїх розмірів, а змінюють лише для віддалення, тобто менше 100%.

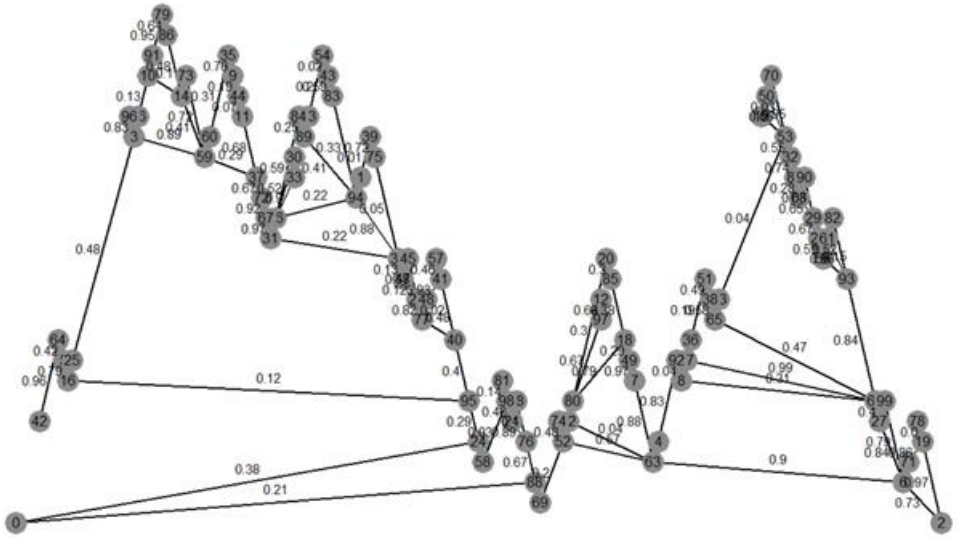


Рисунок 3.8 – Демонстрація масштабування при масштабі 100%

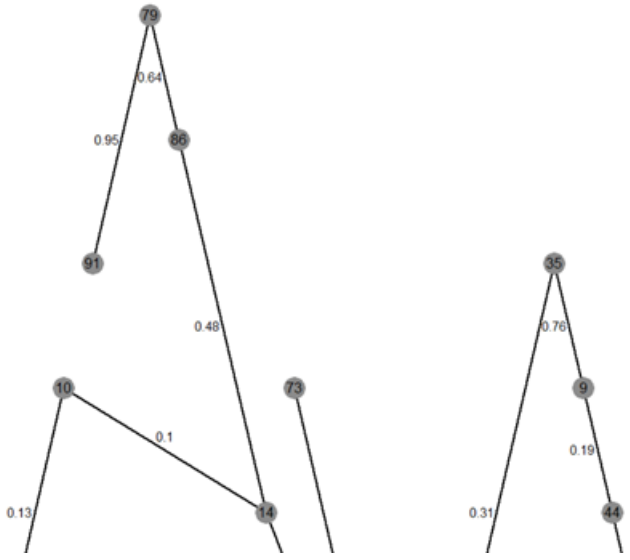


Рисунок 3.9 – Демонстрація масштабування при масштабі 290%

Вікно моделювання створене на основі бібліотеки matplotlib з використанням анімації, для відображення роботи системи, так як основне

вікно є об’єктом бібліотеки, воно має вбудовані функції масштабування та скролінгу. Вікно моделювання має свою функціональну панель, яка складається з кнопки додавання завантаженості, статистики останнього балансування, меню з вибором швидкості відображення анімації (від 1 до 6, де 1 – найшвидший режим, 6 – найповільніший) та кнопки повернення в редактор системи. Вигляд вікна наведено на рис. 3.10.

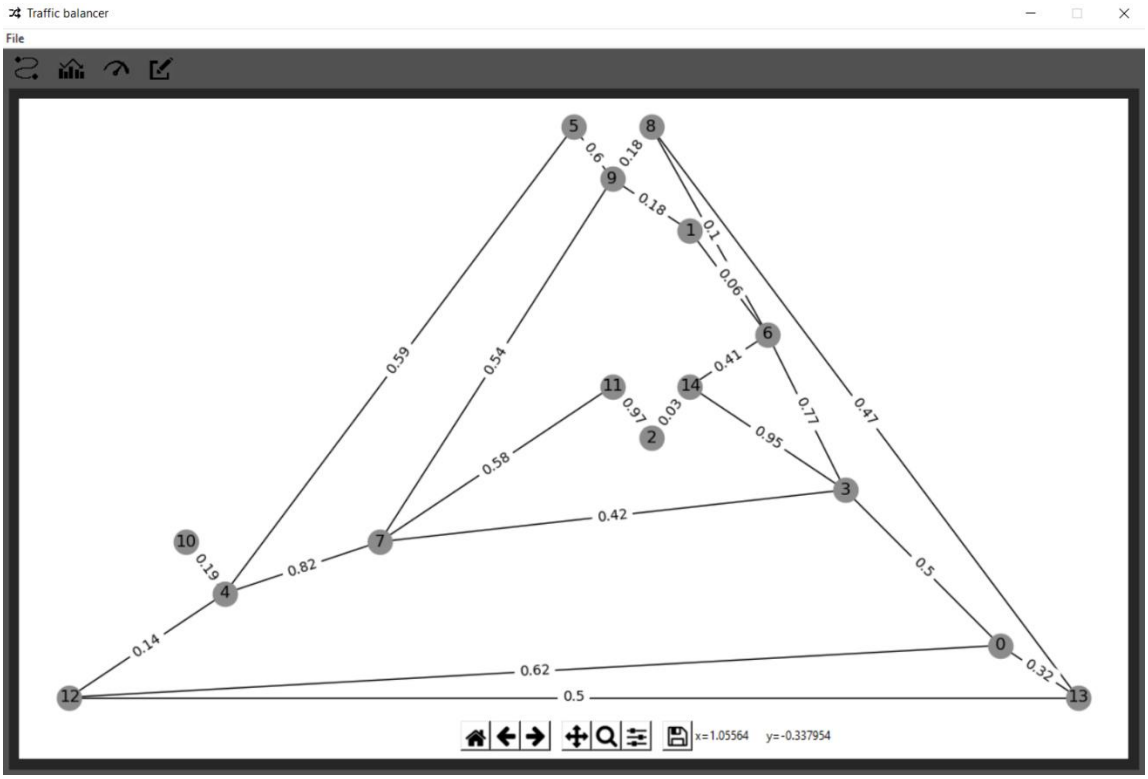


Рисунок 3.10 – Вікно симуляції балансування транспортної системи

Для додання навантаження на систему потрібно натиснути кнопку “додати навантаження” і з’явиться вікно з полями початкової вершини, кінцевої вершини та вагу, яку необхідно збалансувати між цим маршрутом, рис 3.11.

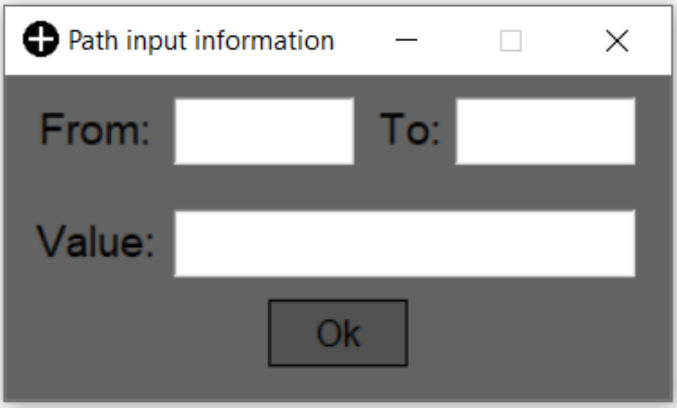


Рисунок 3.11 – Вікно додання завантаженості між обраними вузлами.

Для зміни ваг графу, що відображаються, використовується кнопка правіше від значення масштабування. В нього є 2 параметри – loading, distance. В залежності від обраного режиму відображаються відповідні ваги системи рис.3.12.

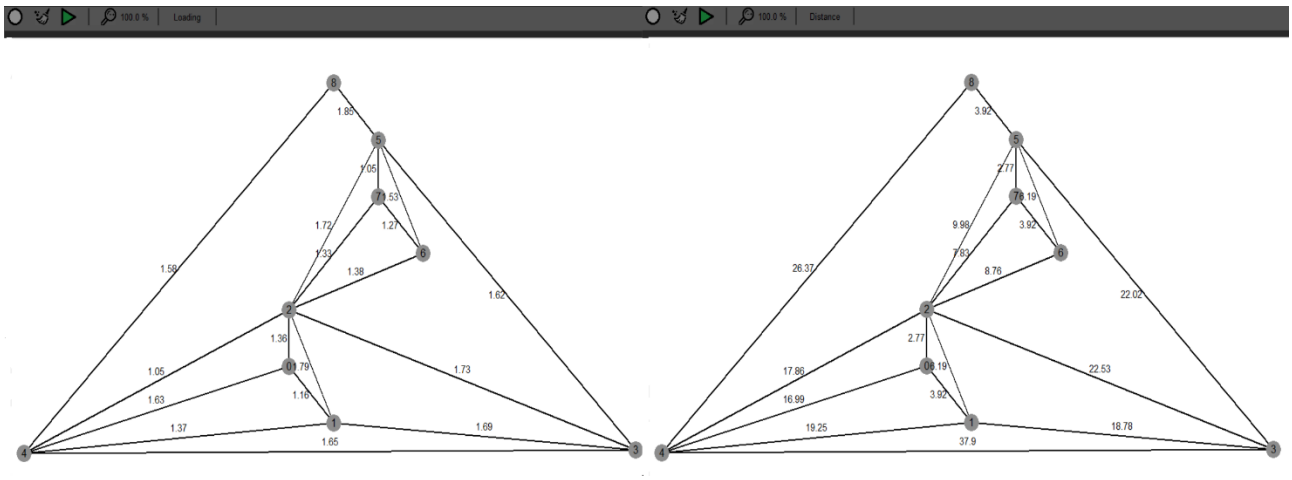


Рисунок 3.12 – Приклад відображення ваг графу від обраного параметру

Для додання навантаження на транспортну систему потрібно перейти у вікно симуляції та натиснути кнопку Find optimal way і з’явиться вікно як на рис 3.11. Після цього вказати початкову та кінцеву вершини шляху в відповідні поля та величину навантаження.

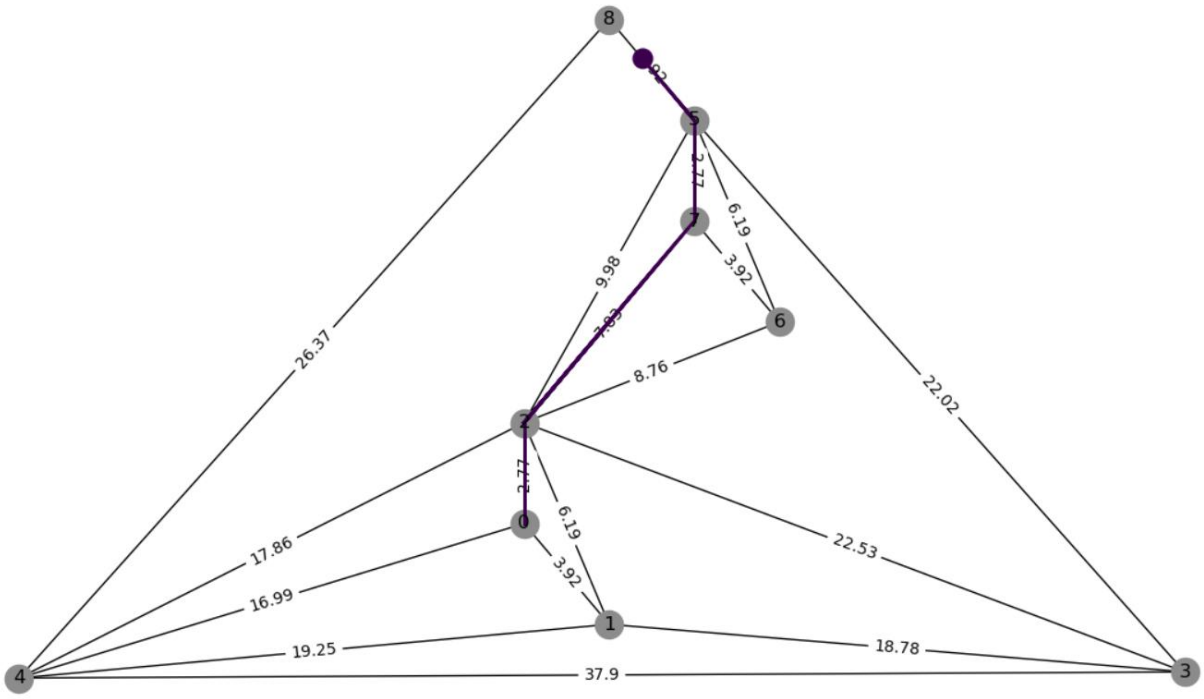


Рисунок 3.13 – Приклад додання навантаження від 0 до 8 вершини

Після додання навантаження можна відкрити вікно статистики натиснувши кнопку Last way details та подивитися деталі балансування (рис 3.14). У разі балансування с динамічним доданням випадкової завантаженості в процесі, шлях може змінитися від початково розрахованого та розраховані їх вартості (рис 3.15).

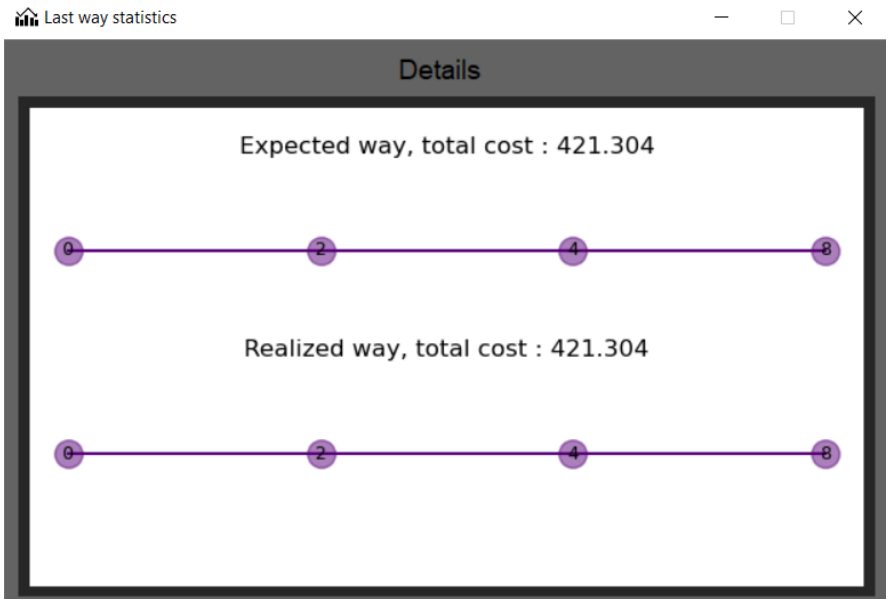


Рисунок 3.14 – Приклад результату балансування без додання випадкової завантаженості

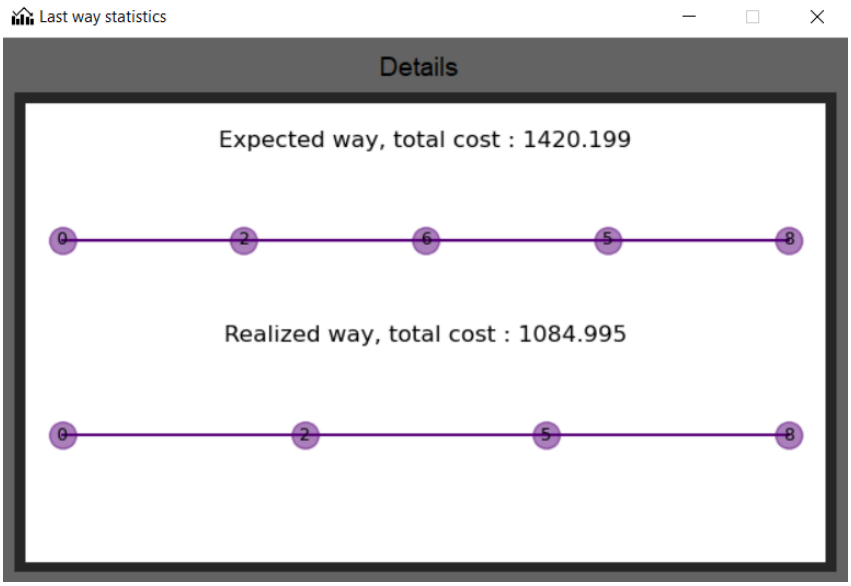


Рисунок 3.15 – Приклад результату балансування з додаванням випадкової завантаженості

Як видно з результатів отриманих на рисунках 3.14 та 3.15, можна побачити ефективність перерахунку шляху, у випадку додання навантаження в режимі реального часу.

Після виконання необхідних дій систему можна зберегти натиснувши file -> save та ввести ім’я файлу. Файл буде збережений у форматі json (рис.3.16). Схема взаємодії програми з користувачем знаходиться в додатку 3.

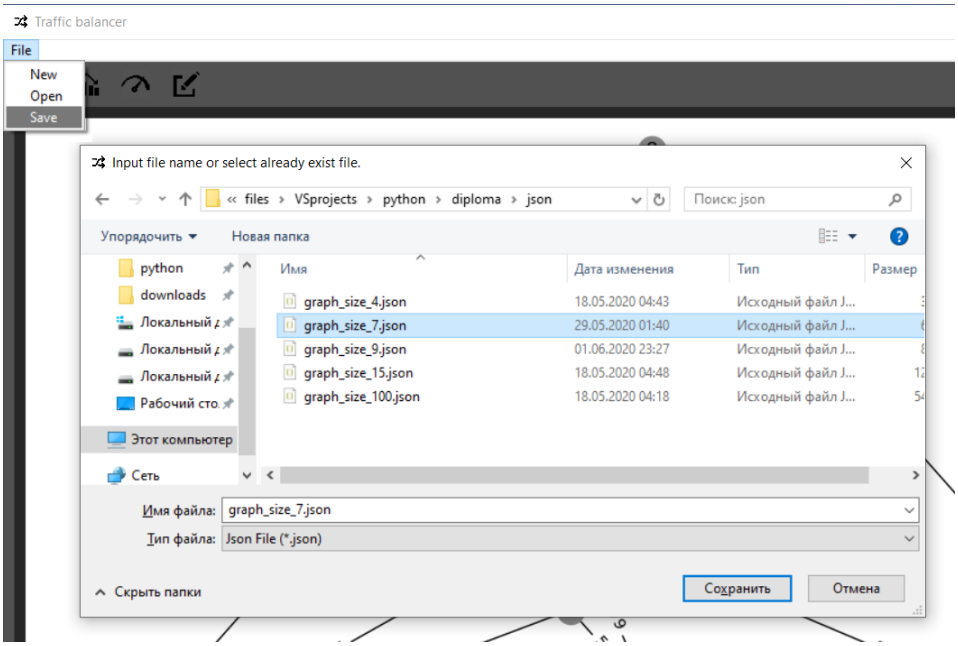


Рисунок 3.16 – Приклад збереження системи в файл

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було розглянуто та обрано технології та інструменти для написання програмного продукту. Був розроблений програмний продукт з графічним інтерфейсом відповідно до поставленої задачі. Зроблений опис класів та детальний розгляд основних методів, реалізованих в програмному продукті. Наведена діаграма класів з відображенням полів кожного класу.

В результаті роботи було отримано доволі непоганий функціонал з реалізованим алгоритмом балансування інтелектуальної транспортної системи та користувацьким інтерфейсом, який легко допомагає взаємодіяти з системою.

ВИСНОВКИ

Даний дипломний проект присвячений створенню програмної реалізації інтелектуальної транспортної системи з застосуванням алгоритму пошуку оптимального шляху з метою балансування трафіку в системі.

У ході роботи було розглянуто транспортні системи, їх види, сфери впливу та роль в життєдіяльності людини. Проведено аналіз актуальності обраної теми та виявлені потреби модернізації транспортних систем. Виділено основні особливості для ефективної організації інтелектуальної транспортної системи.

Було проведено детальний огляд переваг та недоліків існуючих алгоритмів пошуку оптимального шляху, та обраний алгоритм з урахуванням особливостей поставленої задачі.

Також в дипломній роботі було описано обґрунтовування вибору стеку технологій для розробки програми, виділено основні переваги обраного стеку та розгляд застосованих бібліотек.

Було створено програмний продукт відповідно до поставленої мети, з графічним інтерфейсом для зручної взаємодії користувача з підтримкою вказаного функціоналу. Наведено детальний опис особливостей на прикладі її використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Transport system [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/topics/earth-and-planetary-sciences/transport-system (дата звернення 23.01.2020).
2. Transportation system [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/topics/earth-and-planetary-sciences/transportation-system (дата звернення 23.01.2020).
3. Modeling and Control of a New Narrow Vehicle [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/pii/B978012397199900001X (дата звернення 24.01.2020).
4. Evaluation of Intelligent Road Transport Systems: Methods and Results, Lu, Meng, 2016, p. 35-37.
5. Intelligent Transport Systems [Електронний ресурс] – Режим доступу: https://www.unece.org/trans/themes/transtheme-its/intelligent-transport-systems.html (дата звернення 03.02.2020).
6. Введение в теорию графов, Робин Дж. Уилсон, Диалектика-Вильямс, 2020.
7. Генри С. Уоррен мл. – Алгоритмические трюки для программистов, 2014.
8. Introduction to Algorithms, 3rd Edition by Thomas H. Corman, The MIT Press, July 31, 2009, p. 213 – 221.
9. A search algorithm [Електронний ресурс] – Режим доступу: https://www.edureka.co/blog/a-search-algorithm/ (дата звернення 18.02.2020).
10. Fluent Python: Clear, Concise, and Effective Programming 1st Edition, Luciano Ramalho, 2015, p. 94.
11. Python Documentation [Електронний ресурс] – режим доступу: https://www.python.org/ (дата звернення 25.03.2020).
12. Python code style standarts [Електронний ресурс] – режим доступу: https://www.python.org/dev/peps/ (дата звернення 20.04.2020).

13. Python Programming: An Introduction to Computer Science by John Zelle August 2010. p. 122 - 124.

14. Tkinter Documentation [Електронний ресурс] – режим доступу: https://docs.python.org/3/library/tkinter.html (дата звернення 20.03.2020).

15. PMW Documentation [Електронний ресурс] – режим доступу: http://pmw.sourceforge.net/doc/index.html (дата звернення 28.03.2020).

16. Matplotlib Documentation [Електронний ресурс] – режим доступу: https://matplotlib.org/3.2.1/contents.html (дата звернення 11.04.2020).

17. Networkx Documentation [Електронний ресурс] – режим доступу: https://networkx.github.io/documentation/stable/ (дата звернення 14.04.2020).

18. Numpy Documentation [Електронний ресурс] – режим доступу: https://numpy.org/devdocs/docs/howto_document.html (дата звернення 11.04.2020).

19. The Python Tutorial [Електронний ресурс] – Режим доступу: https://docs.python.org/3/tutorial (дата звернення 17.04.2020).

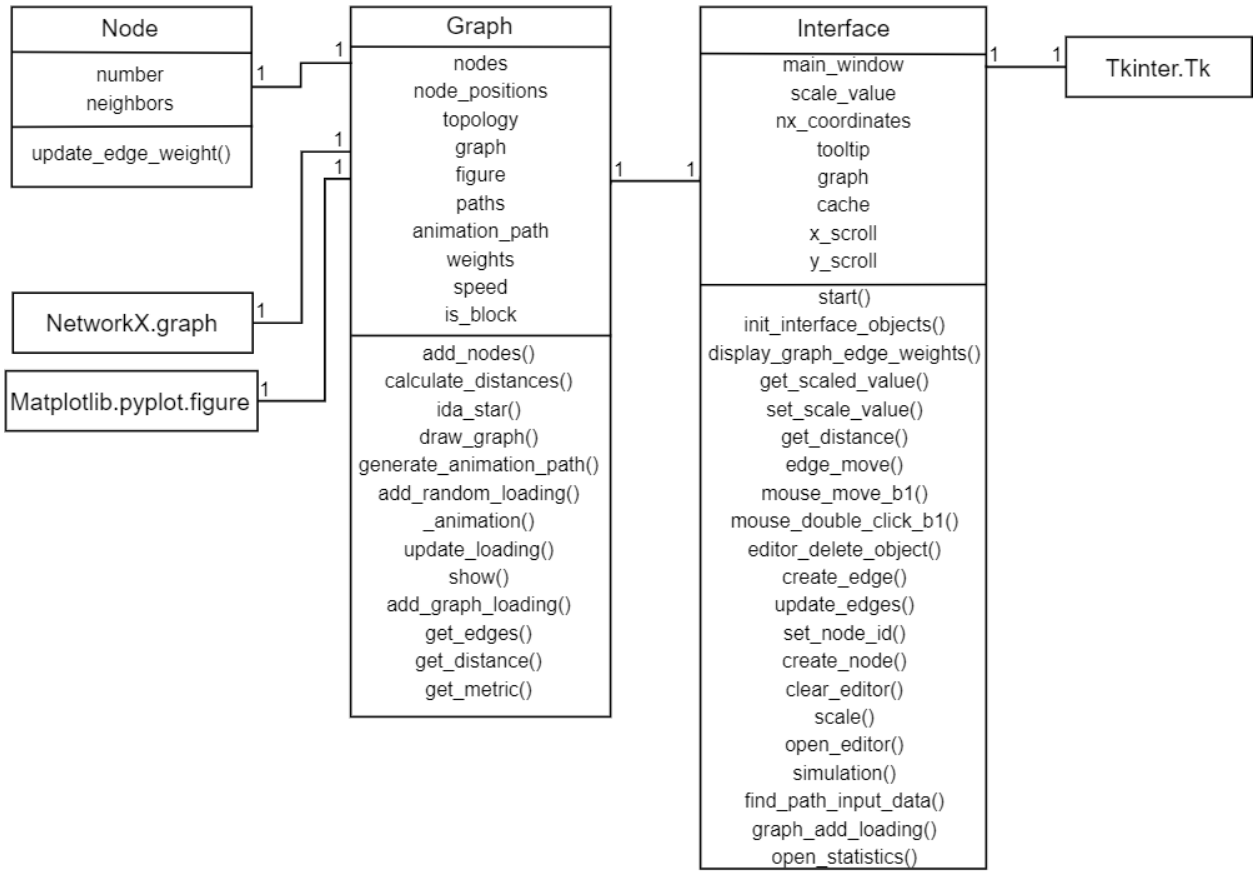
20. Programming Python: Powerful Object-Oriented Programming Fourth Edition by Mark Lutz, January 18, 2011, p. 91, 142.

Додаток 1

до дипломного проєкту

на тему: «Алгоритм балансування трафіку в інтелектуальних транспортних системах»

Київ – 2020 року



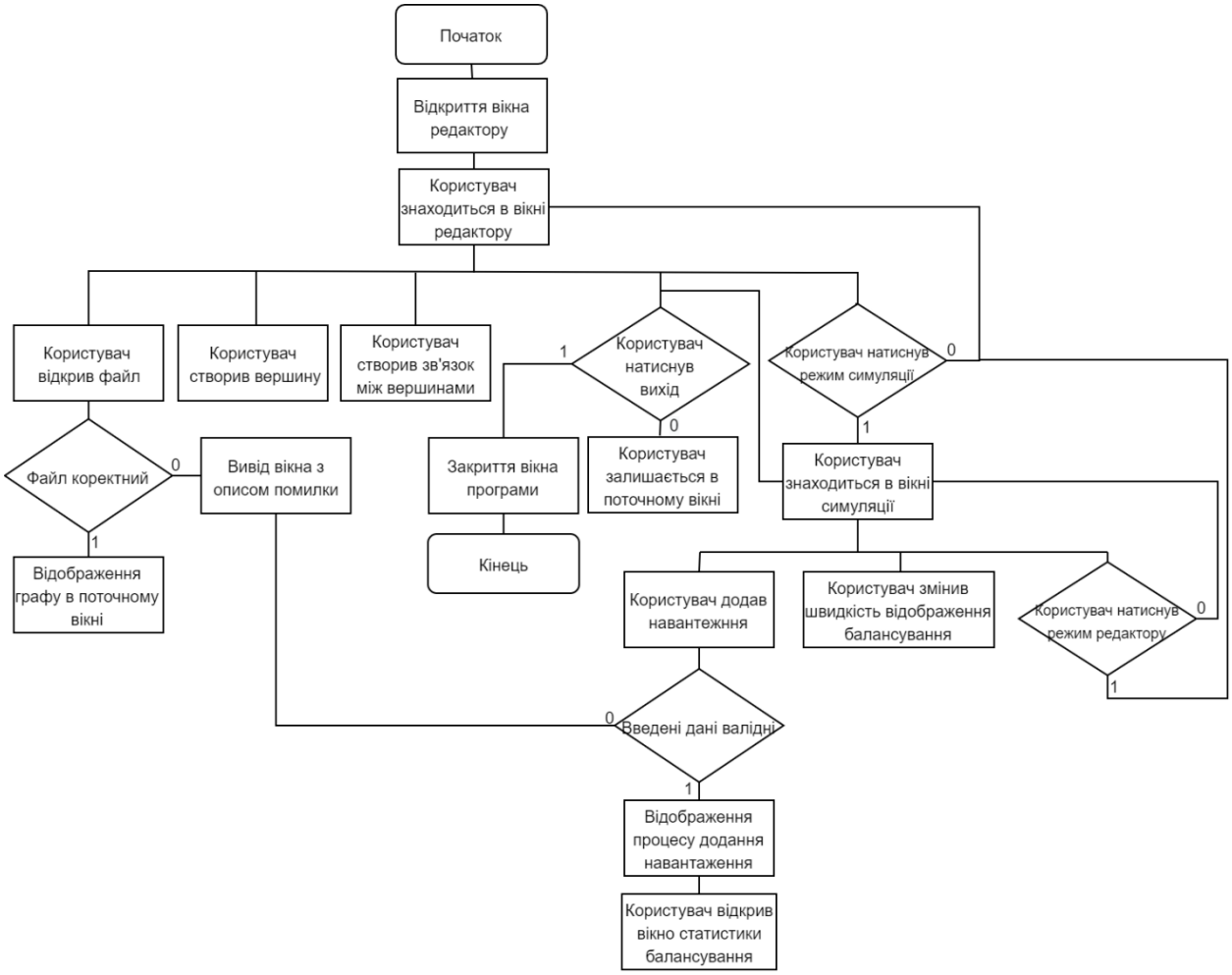
					ІАЛЦ. 467800.004 Д1			
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розробив		*Шапка О.В.*			*Алгоритм балансування трафіку в інтелектуальних транспортних системах*	*Літ.*	*Аркуш*	*Аркушів*
Перевір.							1	1
					Схема класів	*НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62*		
Н. контр.		*Сімоненко В.П.*						
Затверд.								

Додаток 2

до дипломного проєкту

на тему: «Алгоритм балансування трафіку в інтелектуальних транспортних системах»

Київ – 2020 року



					ІАЛЦ. 467800.005 Д2	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розробив		*Шапка О. В.*			*Алгоритм балансування трафіку в інтелектуальних транспортних системах*	*Літ.* / *Аркуш* / *Аркушів*
Перевір.						1 / 1
Н. контр.		*Сімоненко В.П.*			*Алгоритм роботи програми*	*НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62*
Затверд.						

Додаток 3

до дипломного проєкту

на тему: «Алгоритм балансування трафіку в інтелектуальних транспортних системах»

Київ – 2020 року

					ІАЛЦ. 467800.006 Д3			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Шапка О.В.			Алгоритм балансування трафіку в інтелектуальних транспортних системах	Літ.	Аркуш	Аркушів
Перевір.							1	1
					Схема взаємодії с користувачем	НТУУ “КПІ ім. Ігоря Сікорського”, ФІОТ, ІО-62		
Н. контр.		Сімоненко В.П.						
Затверд.								

Додаток 4

до дипломного проєкту

на тему: «Алгоритм балансування трафіку в інтелектуальних транспортних системах»

Київ – 2020 року

ТЕКСТ ПРОГРАМИ

Main.py

```
from tkinter import Tk
from interface import Interface

root = Tk()
gui = Interface(root)
root.mainloop()
```

Graph.py

```
import random
import math

import matplotlib.pyplot as plt
from matplotlib import animation
import networkx as nx
import numpy as np

class Node:
    def __init__(self, number: int, neighbours: list):
        self.number = number
        self.neighbours = neighbours

    def update_edge_weight(self, next_node: int, value: float):
        for i in range(len(self.neighbours)):
            if self.neighbours[i][0] == next_node:
                self.neighbours[i][1] = value

class Graph:
    def __init__(self):
        self.nodes = dict()
        self.node_positions = dict()
        self.topology = dict()
        self.graph = nx.Graph()
        self.figure = plt.figure(figsize=(11.1, 6.6))
        self.paths = dict()
        self.animation_path = list()
        self.weights = str()
        self.speed = 0.05
        self.block = False

    def clear(self):
        self.nodes.clear()
        self.node_positions.clear()
        self.topology.clear()
        self.figure.clear()
        self.graph.clear()
        self.paths.clear()
        self.animation_path.clear()

    def get_distance(self, first_node: int, second_node: int):
        x, y = self.node_positions[first_node]
        x0, y0 = self.node_positions[second_node]
        return math.sqrt(pow(x - x0, 2) + pow(y - y0, 2))

    def get_metric(self):
        edges = list(self.graph.edges(data=True))
        if edges:
            graph_distance = self.get_distance(edges[0][0], edges[0][1])
            return round(edges[0][2]['distance'] / graph_distance, 2)

    def add_nodes(self, positions: dict, nodes: list, weights: str):
        if self.topology:
            self.clear()
        self.weights = weights
        self.node_positions = positions
```

```
        self.nodes = {v.number: v for v in nodes}
        self.topology = {i.number: [] for i in nodes}
        self.create_topology()

        self.graph_plot = self.figure.add_subplot(111)
        plt.subplots_adjust(wspace=0, hspace=1, bottom=0.05, top=1, left=0, right=1)
        self.draw_graph()

    def calculate_distances(self, start: int) -> dict:
        return {node: self.get_distance(start, node)*self.get_metric() for node, coord in self
.node_positions.items()}

    def ida_star(self, start: int, goal: int):
        print('------')
        geometric_distances = self.calculate_distances(goal)

        def search(path, g, bound, types):
            node = path[-1]
            f = g + geometric_distances[node]
            print('f: %s, current bound: %s, path: %s' % (round(f, 3), round(bound, 3), path))
            if f > bound:
                return f
            if node == goal:
                return 'found'
            minn = float('Inf')
            for neighbour in self.nodes[node].neighbours:
                if neighbour[0] not in path:
                    path.append(neighbour[0])
                    t = search(path, g + neighbour[1] * neighbour[2], bound, 'cycle')
                    if t == 'found':
                        return 'found'
                    if t < minn:
                        minn = t
                    path.pop()
            return minn

        path, bound = [start], geometric_distances[start]

        while True:
            t = search(path, 0, bound, 'while')
            if not isinstance(t, str):
                print('path: %s, new bound: %s' % (path, round(t, 3)))
            if t == "found":
                return path
            elif t == float('Inf'):
                return "not-found"
            else:
                bound = t

    def create_topology(self):
        for i in self.nodes.values():
            for j in i.neighbours:
                self.topology[i.number].append(j[0])
                self.graph.add_edge(i.number, j[0], loading=j[1], distance=j[2])

    def draw_graph(self):
        nx.draw(self.graph, pos=self.node_positions, with_labels=True, ax=self.graph_plot, nod
e_color="#8c8c8c")
        nx.draw_networkx_edge_labels(self.graph, pos=self.node_positions, ax=self.graph_plot,
                                     edge_labels=nx.get_edge_attributes(self.graph, self.weigh
ts))

    def generate_animation_path(self, start: int, goal: int) -> list:
        parts = math.ceil(self.get_distance(start, goal) / self.speed)
        self.line_start_coordinates = self.node_positions[start]
        return [np.linspace(self.node_positions[start][0], self.node_positions[goal][0], num=p
arts).tolist(),
                np.linspace(self.node_positions[start][1], self.node_positions[goal][1], num=p
arts).tolist()]
```

```
    def get_edges(self) -> list:
        return list(self.graph.edges(data=True))

    def add_random_loading(self):
        if random.random() > 0.15:
            edges = self.get_edges()
            for edge in random.sample(edges, random.randint(0, math.floor(len(edges)))):
                value = round(self.graph[edge[0]][edge[1]]['loading'] + random.randint(5, 10),
 2)
                self.graph[edge[0]][edge[1]]['loading'] = value
                self.nodes[edge[0]].update_edge_weight(edge[1], value)
                self.nodes[edge[1]].update_edge_weight(edge[0], value)
            nx.draw_networkx_edge_labels(self.graph, pos=self.node_positions, ax=self.graph_pl
ot,
                                         edge_labels=nx.get_edge_attributes(self.graph, self.w
eights))

    def get_coef(self, start: int, goal: int) -> float:
        return self.graph[start][goal]['loading'] * self.graph[start][goal]['distance']

    def _animation(self, interface, with_random: bool, value: float):
        if not self.animation_path:
            if self.start != self.goal:
                self.paths = self.ida_star(self.start, self.goal)
                if self.paths == 'not-found':
                    self.animation.event_source.stop()
                    self.graph_plot.clear()
                    self.draw_graph()
                    self.add_loading_result, self.block = False, False
                    return False

                self.animation_path = self.generate_animation_path(self.start, self.paths[1])
                self.result['result']['path'].append(self.paths[1])
                self.result['result']['cost'] += self.get_coef(self.start, self.paths[1])

                if len(self.result['result']['path']) <= len(self.result['expected']['path']):
                    bound = len(self.result['result']['path'])
                    self.result['expected']['cost'] += self.get_coef(*self.result['expected'][
'path'][(bound - 2):bound])
                    self.result['expected']['count'] += 1

                if with_random:
                    self.add_random_loading()
                self.update_loading(self.start, self.paths[1], value)
            else:
                self.animation.event_source.stop()
                self.graph_plot.clear()
                self.draw_graph()
                self.update_expected_path()
                self.add_loading_result, self.block = True, False
                return True

        if len(self.animation_path[0]) != 0:
            new_x, new_y = self.animation_path[0].pop(0), self.animation_path[1].pop(0)
            self.moving_node.set_data(new_x, new_y)
            self.graph_plot.plot([self.line_start_coordinates[0], new_x], [self.line_start_coo
rdinates[1], new_y],
                                 color='#3d004f', linewidth=2)
        else:
            self.start = self.paths[1]
            self.animation_path.clear()

    def update_loading(self, start: int, goal: int, value: float):
        new_value = round(self.graph[start][goal]['loading'] + value, 2)
        self.graph[start][goal]['loading'] = new_value
        self.nodes[start].update_edge_weight(goal, new_value)
        self.nodes[goal].update_edge_weight(start, new_value)
```

```
    def update_expected_path(self):
        expected = self.result['expected']
        if expected['count'] != len(expected['path']) - 1:
            for i in range(expected['count'] - 1, len(expected['path']) - 1):
                self.result['expected']['cost'] += self.get_coef(expected['path'][i], expected
['path'][i + 1])

    def show(self, value: float, with_random: bool):
        self.moving_node, = self.graph_plot.plot([], [], 'o', color='#3d004f', markersize=12)
        self.animation = animation.FuncAnimation(self.figure, self._animation, frames=200, int
erval=20,
                                                 fargs=(with_random, value,))

    def add_graph_loading(self, start: int, goal: int, value: float, with_random: bool) -
> bool:
        if (start not in self.nodes.keys()) or (goal not in self.nodes.keys()):
            return False

        self.start, self.goal, self.block = start, goal, True
        self.result = {
            'result': {'path': [start], 'cost': 0},
            'expected': {'path': self.ida_star(start, goal), 'cost': 0, 'count': 0}
        }

        self.show(value, with_random)
        return True

    def collect_data(self) -> dict:
        data = {'nodes': []}
        for node in self.topology:
            one_node = {"node_id": node, "connections": []}
            for con_node in self.topology[node]:
                one_node["connections"].append({
                    "node_id": con_node,
                    "loading": self.graph.get_edge_data(node, con_node, 'loading')['loading'],
                    "distance": self.graph.get_edge_data(node, con_node, 'distance')['distance
']})
            data['nodes'].append(one_node)
        data['positions'] = self.node_positions
        return data
```

Interface.py

```
import warnings
import os
import json
import math

from tkinter import Menu, Frame, Canvas, Button, Menubutton, PhotoImage, Label, Entry, \
    Toplevel, messagebox, simpledialog, Scrollbar, BooleanVar, Checkbutton
from tkinter.filedialog import askopenfilename, asksaveasfile
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
from matplotlib import gridspec
import matplotlib.pyplot as plt
import matplotlib
from Pmw import Balloon

from graph import Graph, Node

warnings.simplefilter(action='ignore', category=FutureWarning)
matplotlib.use("Agg")

class Interface:
    def __init__(self, root):
        self.main_window = root
        self.tooltip = Balloon(root)
        self.main_window.geometry("1150x750+100+20")
        self.main_window.resizable(False, False)
        self.main_window.iconbitmap('icons/icon.ico')
```

```
        self.main_window.title('Traffic balancer')
        self.main_window['bg'] = '#515151'
        self.main_window.bind('<Escape>', lambda *ignore: self.main_window.destroy())
        self.main_window.bind('<Delete>', self.editor_delete_object)
        self.graph = Graph()
        self.cache = {'frame': None, 'now_selected': None, 'pair': list(),
                      'nodes': list(), 'edges': dict(), 'last_way_details': None}
        self.scale_value = 1
        self.x_scroll = 1090
        self.y_scroll = 640
        self.nx_coordinates = dict()
        self.start()

    def start(self):
        self.init_interface_objects()
        self.buttons_frame.place(x=0, y=0)
        self.editor_frame.place(x=10, y=40)
        self.toolbar.place(x=1000, y=1000)
        self.open_editor()

    def init_interface_objects(self):
        self.editor_frame = Frame(self.main_window, highlightthickness=10, highlightbackground
="#262626")
        self.editor_canvas = Canvas(self.editor_frame, width=1090, height=640, bg="#ffffff")

        self.editor_xsb = Scrollbar(self.editor_frame, orient="horizontal", command=self.edito
r_canvas.xview)
        self.editor_ysb = Scrollbar(self.editor_frame, orient="vertical", command=self.editor_
canvas.yview)
        self.editor_canvas.configure(xscrollcommand=self.editor_xsb.set, yscrollcommand=self.e
ditor_ysb.set,
                                     scrollregion=(0, 0, self.x_scroll, self.y_scroll))

        self.editor_xsb.pack(side='bottom', fill='x', expand=True)
        self.editor_ysb.pack(side='right', fill='y', expand=True)
        self.editor_canvas.pack(side='left', fill='both', expand=True)

        self.graphic_frame = Frame(self.main_window, width=1130, height=680, bg="#262626")
        self.buttons_frame = Frame(self.main_window, width=1150, height=40, bg="#515151")

        self.canvas = FigureCanvasTkAgg(self.graph.figure, self.graphic_frame)
        self.toolbar = NavigationToolbar2Tk(self.canvas, self.graphic_frame)
        self.toolbar.config(background="#ffffff")

        for button in self.toolbar.winfo_children():
            button.config(background="#ffffff")

        self.path_icon = PhotoImage(file=r"icons\path.png")
        self.path_button = Button(self.buttons_frame, width=25, height=25, image=self.path_ico
n, bd=0,
                                  highlightthickness=0, cursor="hand2", activebackground='#262
626',
                                  command=self.find_path_input_data)
        self.tooltip.bind(self.path_button, 'Find optimal way')

        self.statistics_icon = PhotoImage(file=r"icons\statistics.png")
        self.statistics_loading_button = Button(self.buttons_frame, width=25, height=25, image
=self.statistics_icon,
                                                highlightthickness=0, cursor="hand2", activeba
ckground='#262626', bd=0,
                                                command=self.open_statistics)
        self.tooltip.bind(self.statistics_loading_button, 'Last way details')

        self.display_speed_icon = PhotoImage(file=r"icons\display_speed.png")
        self.display_speed_button = Menubutton(self.buttons_frame, width=25, height=25, image=
self.display_speed_icon,
                                               highlightthickness=0, bd=0, cursor="hand2", act
ivebackground='#262626')
        self.tooltip.bind(self.display_speed_button, 'Change display speed')
```

```
        self.display_speed_button.menu = Menu(self.display_speed_button, tearoff=0, activeback
ground='#787878',
                                              bg='#ffffff')
        self.display_speed_button["menu"] = self.display_speed_button.menu
        self.display_speed_button.menu.add_command(label="1", command=lambda: self.change_disp
lay_speed(1))
        self.display_speed_button.menu.add_command(label="2", command=lambda: self.change_disp
lay_speed(2))
        self.display_speed_button.menu.add_command(label="3", command=lambda: self.change_disp
lay_speed(3))
        self.display_speed_button.menu.add_command(label="4", command=lambda: self.change_disp
lay_speed(4))
        self.display_speed_button.menu.add_command(label="5", command=lambda: self.change_disp
lay_speed(5))
        self.display_speed_button.menu.add_command(label="6", command=lambda: self.change_disp
lay_speed(6))

        self.editor_icon = PhotoImage(file=r"icons\editor.png")
        self.edit_model_button = Button(self.buttons_frame, width=25, height=25, image=self.ed
itor_icon, bd=0,
                                        highlightthickness=0, cursor="hand2", activebackground
='#262626',
                                        command=self.open_editor)
        self.tooltip.bind(self.edit_model_button, 'Editor')

        self.create_node_icon = PhotoImage(file=r"icons\node.png")
        self.create_node_button = Button(self.buttons_frame, width=25, height=25, image=self.c
reate_node_icon, bd=0,
                                         highlightthickness=0, cursor="hand2", activebackgroun
d='#262626',
                                         command=self.create_node)
        self.tooltip.bind(self.create_node_button, 'Create node')

        self.clear_editor_icon = PhotoImage(file=r"icons\clear.png")
        self.clear_editor_button = Button(self.buttons_frame, width=25, height=25, image=self.
clear_editor_icon, bd=0,
                                          highlightthickness=0, cursor="hand2", activebackgrou
nd='#262626',
                                          command=self.clear_editor)
        self.tooltip.bind(self.clear_editor_button, 'Clear editor')

        self.simulation_icon = PhotoImage(file=r"icons\simulation.png")
        self.simulation_button = Button(self.buttons_frame, width=25, height=25, image=self.si
mulation_icon, bd=0,
                                        highlightthickness=0, cursor="hand2", activebackground
='#262626',
                                        command=self.simulation)
        self.tooltip.bind(self.simulation_button, 'Simulation')

        self.zoom_icon = PhotoImage(file=r"icons\zoom.png")
        self.zoom_frame = Frame(self.buttons_frame, height=25, width=119, bg='#262626')
        self.zoom_image = Label(self.zoom_frame, image=self.zoom_icon, highlightthickness=0, b
d=0)
        self.zoom_text = Label(self.zoom_frame, bg="#515151", highlightthickness=0, bd=0,
                               font=("arial", 10), pady=5, padx=7, width=7, text='100.0 %', an
chor='w')
        self.zoom_text.place(x=47, y=0)
        self.zoom_image.place(x=2, y=0)
        self.tooltip.bind(self.zoom_image, 'Zoom')

        self.change_edge_weight_frame = Frame(self.buttons_frame, height=25, width=96, bg='#26
2626')
        self.change_edge_weight_button = Menubutton(self.change_edge_weight_frame, text='Loadi
ng', font=("arial", 10),
                                                    width=12, pady=5, padx=5,  bg="#515151", b
d=0, cursor="hand2",
                                                    activebackground='#787878', highlightthick
ness=0)
```

```
        self.tooltip.bind(self.change_edge_weight_button, 'Change displaying edge weights')
        self.change_edge_weight_button.place(x=0, y=0)

        self.change_edge_weight_button.menu = Menu(self.change_edge_weight_button, tearoff=0,
font=("arial", 10),
                                                   bg='#ffffff', activebackground='#787878')
        self.change_edge_weight_button["menu"] = self.change_edge_weight_button.menu
        self.change_edge_weight_button.menu.add_command(label="Loading",
                                                        command=lambda: self.display_graph_edg
e_weights("Loading"))
        self.change_edge_weight_button.menu.add_command(label="Distance",
                                                        command=lambda: self.display_graph_edg
e_weights("Distance"))
        self.menu_bar = Menu(self.main_window)
        self.main_window['menu'] = self.menu_bar
        self.file_menu = Menu(self.menu_bar, tearoff=0, bg='#ffffff', activebackground='#78787
8')
        self.menu_bar.add_cascade(label="File", menu=self.file_menu)
        self.file_menu.add_command(label="New", command=self.new_file)
        self.file_menu.add_command(label="Open", command=self.open_file)
        self.file_menu.add_command(label="Save", command=self.save_file)

    def display_graph_edge_weights(self, weight: str):
        if self.change_edge_weight_button['text'] != weight:
            self.change_edge_weight_button.config(text=weight)
            for obj in self.editor_canvas.find_all():
                obj_tag = self.editor_canvas.gettags(obj)[0]
                if obj_tag.startswith('linelabel'):
                    edge_key = tuple(sorted(list(map(int, obj_tag.split('_')[1:]))))
                    self.editor_canvas.itemconfigure(obj_tag, text=self.cache['edges'][edge_ke
y][weight.lower()])

                    edge_text, coord = str(self.editor_canvas.itemcget(obj_tag, 'text')), \
                        self.editor_canvas.coords(obj_tag.replace('linelabel', 'line'))
                    self.editor_canvas.coords(obj_tag, self.get_egde_text_position(str(edge_te
xt), *coord))

    def get_scaled_value(self, value: int):
        step = 1 if self.scale_value >= 1 else -1
        scale_coef = 1.1 if self.scale_value >= 1 else 0.9
        for _ in range(10, int(self.scale_value * 10), step):
            value = value * scale_coef
        return value

    def set_scroll_range(self, x_lim: int, y_lim: int):
        self.editor_canvas.configure(scrollregion=(0, 0, x_lim, y_lim))

    def set_scale_value(self, value: float, set_only_text: bool = False):
        if not set_only_text:
            self.scale_value = value
        self.set_scroll_range(self.get_scaled_value(self.x_scroll), self.get_scaled_value(self
.y_scroll))
        self.zoom_text.configure(text="{:.1f} %".format(self.scale_value * 100))

    def get_distance(self, coords):
        x, y, x0, y0 = coords
        if hasattr(self, 'xmul'):
            difference = self.xmul / self.ymul
        else:
            difference = 1
        return math.sqrt(pow((x-x0), 2) + pow(((y-y0) * difference), 2))

    def recalculate_distance(self, edge_key: tuple, old_coords: tuple, new_coords: tuple) -
> float:
        distance_per_pixel = self.cache['edges'][edge_key]['distance'] / self.get_distance(old
_coords)
        return round(self.get_distance(new_coords) * distance_per_pixel, 2)

    def edge_move(self, node_id: int, x: int, y: int):
```

```
        lines_tags = [i for i in list(self.cache['edges'].keys()) if node_id in i]
        lines_coords = [self.editor_canvas.coords('line_%s_%s' % (i[0], i[1])) for i in lines_
tags]
        for i in range(len(lines_tags)):
            if lines_tags[i][0] == node_id:
                new_coords = [x, y, lines_coords[i][2], lines_coords[i][3]]
            else:
                new_coords = [lines_coords[i][0], lines_coords[i][1], x, y]
            self.editor_canvas.coords('line_%s_%s' % lines_tags[i], new_coords)
            new_distance = self.recalculate_distance(lines_tags[i], lines_coords[i], new_coord
s)
            self.cache['edges'][lines_tags[i]]['distance'] = new_distance

            if self.change_edge_weight_button['text'].lower() == 'loading':
                edge_text = str(self.editor_canvas.itemcget('linelabel_%s_%s' % lines_tags[i],
 'text'))
            else:
                edge_text = str(new_distance)
                self.editor_canvas.itemconfigure('linelabel_%s_%s' % lines_tags[i], text=edge_
text)
            self.editor_canvas.coords('linelabel_%s_%s' % lines_tags[i], self.get_egde_text_po
sition(edge_text,

       *new_coords))

    def mouse_move_b1(self, event):
        node_tag = self.editor_canvas.gettags(event.widget.find_withtag('current'))
        if node_tag:
            if 'node' in node_tag[0] or 'text' in node_tag[0]:
                node_id = int(node_tag[0].split('_')[1])
                x1, y1, x2, y2 = self.editor_canvas.coords('node_%s' % (node_id))
                if ((x1 > 12 and y1 > 12) and (x2 < self.x_scroll-12 and y2 < self.y_scroll-
12)) or \
                   ((event.x > 12 and event.y > 12) and (event.x < 1078 and event.y < 628)):
                    scrollregion = list(map(float, self.editor_canvas['scrollregion'].split('
')))
                    sx, sy = self.editor_canvas.xview()[0] * scrollregion[2], \
                        self.editor_canvas.yview()[0] * scrollregion[3]
                    half = 12 if self.scale_value > 1 else self.get_scaled_value(12)
                    self.edge_move(node_id, event.x + sx, event.y + sy)
                    self.editor_canvas.coords('node_%s' % (node_id),
                                              event.x-half+sx, event.y-half+sy,
                                              event.x+half+sx, event.y+half+sy)
                    self.editor_canvas.coords('text_%s' % (node_id),
                                              event.x+sx, event.y+sy)

    def mouse_double_click_b1(self, event):
        tag = self.editor_canvas.gettags(event.widget.find_withtag('current'))
        if tag:
            if tag[0].startswith('node_') or tag[0].startswith('text_'):
                node_id = int(tag[0].split('_')[1])
                if self.cache['now_selected']:
                    if self.cache['now_selected']['type'] == 'edge':
                        self.editor_canvas.itemconfigure(self.cache['now_selected']['tag'], fi
ll='#000000')
                if len(self.cache['pair']) < 2:
                    if node_id not in self.cache['pair']:
                        self.cache['now_selected'] = {'type': 'node', 'tag': tag[0].replace('t
ext', 'node')}
                        self.editor_canvas.itemconfigure("node_%s" % node_id, fill='#cccccc')
                        self.cache['pair'].append(node_id)
                    else:
                        self.cache['now_selected'] = None
                        self.editor_canvas.itemconfigure("node_%s" % node_id, fill='#8c8c8c')
                        self.cache['pair'].remove(node_id)

                if len(self.cache['pair']) == 2:
                    self.cache['pair'].sort()
```

```
                    first_tag, second_tag = "node_%s" % self.cache['pair'][0], "node_%s" % sel
f.cache['pair'][1]

                    if tuple(self.cache['pair']) not in self.cache['edges']:
                        answer_load = simpledialog.askfloat("Input", "Input edge loading.", pa
rent=self.main_window,
                                                            minvalue=0.0, maxvalue=10000)
                        if answer_load:
                            pair = self.cache['pair']
                            xn1, yn1, x1n1, y1n1 = tuple(self.editor_canvas.coords('node_%s' %
 pair[0]))
                            xn2, yn2, x1n2, y1n2 = tuple(self.editor_canvas.coords('node_%s' %
 pair[1]))
                            new_edge = tuple([(xn1+x1n1)/2, (yn1+y1n1)/2, (xn2+x1n2)/2, (yn2+y
1n2)/2])

                            if self.cache['edges']:
                                edge = tuple(list(self.cache['edges'].keys())[0])
                                distance = self.recalculate_distance(edge,
                                                                     self.editor_canvas.coords
('line_%s_%s' % edge),
                                                                     new_edge)
                            else:
                                distance = self.get_distance(new_edge) * 2

                            self.create_edge(tuple(self.cache['pair']), {
                                'loading': round(answer_load, 2),
                                'distance': distance
                            })
                    else:
                        messagebox.showinfo('Editor error', 'These nodes are already connected
.')

                    self.cache['pair'].clear()
                    self.cache['now_selected'] = None
                    self.editor_canvas.itemconfigure(first_tag, fill='#8c8c8c')
                    self.editor_canvas.itemconfigure(second_tag, fill='#8c8c8c')

            elif tag[0].startswith('linelabel_'):
                if self.change_edge_weight_button['text'].lower() == 'loading':
                    value = simpledialog.askfloat("Input", f"Change edge loading.", parent=sel
f.main_window,
                                                  minvalue=0.0, maxvalue=10000)
                    if value:
                        edge_key, value = tuple(sorted(list(map(int, tag[0].split('_')[1:])))
, round(value, 2)
                        self.cache['edges'][edge_key]['loading'] = value
                        self.editor_canvas.itemconfigure(tag[0], text=value)

            elif tag[0].startswith('line_'):
                cache_info = {'type': 'edge', 'tag': tag[0]}

                if self.cache['now_selected'] == cache_info:
                    self.cache['now_selected'] = None
                    self.editor_canvas.itemconfigure(tag[0], fill='#000000')
                else:
                    if self.cache['now_selected']:
                        if self.cache['now_selected']['type'] == 'edge':
                            self.editor_canvas.itemconfigure(self.cache['now_selected']['tag']
, fill='#000000')
                        elif self.cache['now_selected']['type'] == 'node':
                            self.editor_canvas.itemconfigure(self.cache['now_selected']['tag']
, fill='#8c8c8c')
                            self.cache['pair'].clear()
                    self.cache['now_selected'] = cache_info
                    self.editor_canvas.itemconfigure(tag[0], fill='#b5b5b5')

    def editor_delete_object(self, event):
        if self.cache['now_selected'] is not None:
```

```
            if self.cache['now_selected']['type'] == 'node':
                node_id = int(self.cache['now_selected']['tag'].split('_')[1])
                self.editor_canvas.delete('node_%s' % node_id)
                self.editor_canvas.delete('text_%s' % node_id)
                self.cache['nodes'].remove(node_id)
                for key in list(self.cache['edges']):
                    if node_id in key:
                        self.editor_canvas.delete('line_%s_%s' % (key))
                        self.editor_canvas.delete('linelabel_%s_%s' % (key))
                        del self.cache['edges'][key]
                if node_id in self.cache['pair']:
                    self.cache['pair'].remove(node_id)
                self.cache['now_selected'] = None

            elif self.cache['now_selected']['type'] == 'edge':
                self.editor_canvas.delete(self.cache['now_selected']['tag'])
                self.editor_canvas.delete(self.cache['now_selected']['tag'].replace('line', 'l
inelabel'))
                del self.cache['edges'][tuple(sorted(list(map(int, self.cache['now_selected'][
'tag'].split('_')[1:]))))]
                self.cache['now_selected'] = None

    def get_egde_text_position(self, text: str, x1: float, y1: float, x2: float, y2: float) -
> tuple:
        new_x, new_y = (x1 + x2) / 2, (y1 + y2) / 2
        if abs(x1 - x2) > abs(y1 - y2):
            new_y -= 15
        else:
            new_x -= ((len(text) - 1) * 5 + 2) if '.' in text else (len(text) * 7)
        return new_x, new_y

    def create_edge(self, node_ids: tuple, weights: dict):
        first = self.editor_canvas.coords('node_%s' % node_ids[0])
        second = self.editor_canvas.coords('node_%s' % node_ids[1])

        line_tag, line_weight_tag = 'line_%s_%s' % node_ids, 'linelabel_%s_%s' % node_ids

        line_coords = (sum(first[::2])/2, sum(first[1::2])/2, sum(second[::2])/2, sum(second[1
::2])/2)

        self.editor_canvas.tag_lower(self.editor_canvas.create_line(*line_coords, width=2, tag
s=line_tag))
        text = weights[self.change_edge_weight_button['text'].lower()]
        self.editor_canvas.create_text(self.get_egde_text_position(str(text), *line_coords), t
ags=line_weight_tag,
                                       font=("Lato", 11), text=text)

        self.cache['edges'][node_ids] = weights

    def update_edges(self):
        for edge in self.graph.get_edges():
            edge_key = tuple(sorted([edge[0], edge[1]]))
            self.editor_canvas.itemconfigure('linelabel_%s_%s' % edge_key, text=edge[2]['loadi
ng'])
            self.cache['edges'][edge_key]['loading'] = edge[2]['loading']

    def set_node_id(self) -> int:
        length = len(self.cache['nodes'])
        if not length:
            self.cache['nodes'].append(0)
            return 0
        else:
            for node_id in range(len(self.cache['nodes'])):
                if node_id not in self.cache['nodes']:
                    self.cache['nodes'].append(node_id)
                    return node_id
            else:
                self.cache['nodes'].append(length)
                return length
```

```
    def create_node(self, coordinates=None):
        tag_id = self.set_node_id()
        if not coordinates:
            coord_1, coord_2, middle = 30, 54, 42
            node_coord = (coord_1, coord_1, coord_2, coord_2)
            text_coord = (middle, middle)
        else:
            node_coord, text_coord = coordinates

        self.editor_canvas.create_oval(*node_coord, fill='#8c8c8c', tags='node_%s' % tag_id, o
utline="")
        self.editor_canvas.create_text(*text_coord, text=tag_id, tags='text_%s' % tag_id, font
=("Lato", 12))

    def clear_editor(self, mode: str = 'user'):
        if mode == 'user':
            answer = messagebox.askquestion('Editor', 'Are you sure to delete all objects?', i
con='warning')
        else:
            answer = 'yes'

        if answer == 'yes':
            self.editor_canvas.delete('all')
            self.cache['now_selected'] = None
            self.cache['pair'] = list()
            self.cache['nodes'] = list()
            self.cache['edges'] = dict()
            self.cache['last_way_details'] = None
            self.graph.clear()
            self.canvas.draw()

    def scale(self, coef: float):
        for item in self.editor_canvas.find_all():
            tag, coords = self.editor_canvas.gettags(item)[0], list(map(lambda d: d * coef,
                                                                        self.editor_canvas.coo
rds(item)))
            if tag.startswith('node') and (self.scale_value > 1):
                mx, my = (coords[0] + coords[2])/2, (coords[1] + coords[3])/2
                self.editor_canvas.coords(item, mx-12, my-12, mx+12, my+12)
            elif tag.startswith('linelabel') and (self.scale_value > 1):
                edge_coord, edge_text = self.editor_canvas.coords(tag.replace('label', '')), \
                    str(self.editor_canvas.itemcget(item, 'text'))
                self.editor_canvas.coords(item, self.get_egde_text_position(edge_text, *edge_c
oord))
            else:
                self.editor_canvas.coords(item, coords)

    def open_editor(self):

        def zoom(event):
            if event.delta > 0:
                if self.scale_value < 3:
                    self.scale(1.1)
                    self.scale_value += 0.1
            else:
                if self.scale_value > 0.2:
                    self.scale(0.90909090)
                    self.scale_value -= 0.1

            self.set_scale_value(self.scale_value, set_only_text=True)

        def scan(event):
            self.editor_canvas.scan_mark(event.x, event.y)

        def move(event):
            if self.scale_value > 1:
                self.editor_canvas.scan_dragto(event.x, event.y, gain=1)
                self.editor_canvas.update()
```

```
        if self.cache['frame'] != 'editor':
            self.editor_canvas.bind('<B1-Motion>', self.mouse_move_b1)
            self.editor_canvas.bind('<Double-Button-1>', self.mouse_double_click_b1)
            self.editor_canvas.bind("<MouseWheel>", zoom)
            self.editor_canvas.bind('<ButtonPress-3>', scan)
            self.editor_canvas.bind("<B3-Motion>", move)

            self.create_node_button.place(x=15, y=7)
            self.clear_editor_button.place(x=60, y=7)
            self.simulation_button.place(x=105, y=7)
            self.zoom_frame.place(x=150, y=7)
            self.change_edge_weight_frame.place(x=270, y=7)

            self.graphic_frame.place_forget()
            self.path_button.place_forget()
            self.statistics_loading_button.place_forget()
            self.display_speed_button.place_forget()
            self.edit_model_button.place_forget()

            self.cache['frame'] = 'editor'

    def editor_save_topology(self):
        topology = {'nodes': list()}
        for i in self.cache['nodes']:
            one_node = {"node_id": i, "connections": list()}
            for con_node in [j for j in self.cache['edges'] if i in j]:
                one_node["connections"].append({"node_id": con_node[0] if con_node.index(i) ==
 1 else con_node[1],
                                                "loading": self.cache['edges'][con_node]['load
ing'],
                                                'distance': self.cache['edges'][con_node]['dis
tance']})
            topology['nodes'].append(one_node)
        topology['positions'] = self.get_nodes_coordinates()
        return topology

    def new_file(self):
        self.clear_editor(mode='new_file')
        if self.cache['frame'] == 'simulation':
            self.open_editor()

    def open_file(self):
        file = askopenfilename(initialdir=os.path.dirname(os.path.abspath(__file__)),
                               filetypes=(("Json File", "*.json"), ("All Files", "*.*")),
                               title="Choose a file.")
        if file:
            try:
                with open(file) as json_file:
                    data = json.load(json_file)
            except Exception:
                messagebox.showinfo('File open error', 'Can`t open file.')
                return

            try:
                if self.cache['nodes']:
                    self.clear_editor(mode='open_file')
                self.set_scale_value(1)

                node_list = [Node(i['node_id'], [list(map(lambda x: round(x, 2),
                             j.values())) for j in i['connections']]) for i in data['nodes']]
                self.nx_coordinates = {
                    int(k): tuple(v)
                    for k, v in sorted(data['positions'].items(), key=lambda x: x[0])
                }
                self.graph.add_nodes(self.nx_coordinates, node_list, self.change_edge_weight_b
utton['text'].lower())
                self.cache['edges'] = {
                    tuple(sorted([k[0], k[1]])): {
```

```
                        'loading': k[2]['loading'],
                        'distance': k[2]['distance']
                        } for k in self.graph.graph.edges.data()
                }
                self.editor_draw_opened_graph()
                self.canvas.draw()
            except Exception as e:
                messagebox.showinfo('File read error', f'Incorrect file data.\nError message:
{str(e)}')
                return

    def editor_draw_opened_graph(self):
        coords = [i for i in self.graph.node_positions.values()]
        minx, maxx,  = min(coords, key=lambda x: x[0])[0], max(coords, key=lambda x: x[0])[0]
        miny, maxy = min(coords, key=lambda y: y[1])[1], max(coords, key=lambda y: y[1])[1]
        self.x_abs_avg, self.y_abs_avg = abs(minx) + abs(maxx), abs(miny) + abs(maxy)
        self.xmul, self.xshift = 1050 / self.x_abs_avg, 540 - ((minx + maxx) * 540) / 2
        self.ymul, self.yshift = 575 / self.y_abs_avg, 315 + (miny + maxy) * 315 * (2 if abs(m
iny + maxy) < 0.15 else 1)

        for _, pos in sorted(self.graph.node_positions.items(), key=lambda x: x[0]):
            x, y = pos[0] * self.xmul + self.xshift, -pos[1] * self.ymul + self.yshift
            self.create_node(coordinates=((x-12, y-12, x+12, y+12), (x, y)))

        for edge in self.cache['edges']:
            self.create_edge(edge, self.cache['edges'][edge])

    def save_file(self):
        file = asksaveasfile(initialdir=os.path.dirname(os.path.abspath(__file__)),
                             filetypes=(("Json File", "*.json"), ("All Files", "*.*")),
                             title="Input file name or select already exist file.", mode='w',
defaultextension=".json")
        if file:
            if self.cache['frame'] == 'editor':
                json.dump(self.editor_save_topology(), file, ensure_ascii=False, indent=4)
            elif self.cache['frame'] == 'simulation':
                json.dump(self.graph.collect_data(), file, ensure_ascii=False, indent=4)

    def get_nodes_coordinates(self) -> dict:
        nodes = {}
        for obj in self.editor_canvas.find_all():
            obj_tag = self.editor_canvas.gettags(obj)[0]
            if obj_tag.startswith('node_'):
                x, y, x1, y1 = self.editor_canvas.coords(obj)
                if hasattr(self, 'xmul'):
                    value = tuple([(x+x1)/2/self.xmul - self.x_abs_avg/2, -
(y+y1)/2/self.ymul + self.y_abs_avg/2])
                else:
                    value = tuple([(x+x1)/1080 - 1, -(y+y1)/640 + 1])
                nodes[int(obj_tag.split('_')[1])] = value
        return nodes

    def update_graph(self):
        if self.cache['frame'] == 'editor':
            if self.cache['nodes']:
                node_list = [Node(i, list()) for i in self.cache['nodes']]
                for e in self.cache['edges']:
                    node_list[self.cache['nodes'].index(e[0])].neighbours.append([e[1],
                                                                                  *self.cache[
'edges'][e].values()])
                    node_list[self.cache['nodes'].index(e[1])].neighbours.append([e[0],
                                                                                  *self.cache[
'edges'][e].values()])
                self.graph.add_nodes(self.get_nodes_coordinates(), node_list,
                                     self.change_edge_weight_button['text'].lower())
                self.canvas.draw()

    def simulation(self):
        self.create_node_button.place_forget()
```

```
        self.clear_editor_button.place_forget()
        self.zoom_frame.place_forget()
        self.change_edge_weight_frame.place_forget()
        self.simulation_button.place_forget()

        self.canvas._tkcanvas.place(x=10, y=10)
        self.toolbar.place(x=450, y=630)
        self.graphic_frame.place(x=10, y=40)
        self.path_button.place(x=15, y=7)
        self.statistics_loading_button.place(x=60, y=7)
        self.display_speed_button.place(x=105, y=7)
        self.edit_model_button.place(x=150, y=7)

        self.update_graph()
        self.cache['frame'] = 'simulation'

    def find_path_input_data(self):
        path_window = Toplevel(self.main_window, bg="#636363", name="find_path_window")
        path_window.geometry('239x165+115+115')
        path_window.resizable(False, False)
        path_window.title("Path input information")
        path_window.iconbitmap('icons/path_input.ico')

        from_node_label = Label(path_window, text="From:", width=5, font=("Lato", 13), bg="#63
6363", fg="#000000")
        from_node_label.place(x=8, y=8)

        to_node_label = Label(path_window, text="To:", width=5, font=("Lato", 13), bg="#636363
", fg="#000000")
        to_node_label.place(x=120, y=8)

        value_input_label = Label(path_window, text="Value:", width=5, font=("Lato", 13), bg="
#636363", fg="#000000")
        value_input_label.place(x=10, y=50)

        self.from_node_entry = Entry(path_window, font=("Lato", 11), bg="#ffffff", fg="#000000
", cursor="hand2",
                                     validate="key", justify='center')
        self.from_node_entry.configure(validatecommand=(self.from_node_entry.register(self.inp
ut_validation),
                                                         '%P', '%d', 'int'))
        self.from_node_entry.place(x=65, y=8, height=27, width=60)

        self.to_node_entry = Entry(path_window, font=("Lato", 11), bg="#ffffff", fg="#000000",
 cursor="hand2",
                                   validate="key", justify='center')
        self.to_node_entry.configure(validatecommand=(self.to_node_entry.register(self.input_v
alidation),
                                                       '%P', '%d', 'int'))
        self.to_node_entry.place(x=165, y=8, height=27, width=60)

        self.value_input_entry = Entry(path_window, font=("Lato", 11), bg="#ffffff", fg="#0000
00", cursor="hand2",
                                       validate="key", justify='center')
        self.value_input_entry.configure(validatecommand=(self.value_input_entry.register(self
.input_validation),
                                                           '%P', '%d', 'float'))
        self.value_input_entry.place(x=65, y=50, height=27, width=160)

        self.with_random = BooleanVar()
        self.with_random.set(0)
        with_random_check = Checkbutton(path_window, text=(" Random loading"), variable=self.w
ith_random,
                                        onvalue=1, offvalue=0, bg="#636363", fg="#000000", act
ivebackground="#636363",
                                        font=("Lato", 12))
        with_random_check.place(x=0, y=87, width=239)
```

```
        accept_loading_button = Button(path_window, text="Ok", width=6, bg="#525252", fg="#000
000", relief='solid',
                                       borderwidth=1, highlightbackground="#262626", font=("La
to", 12), cursor="hand2",
                                       activebackground="#737373", command=self.graph_add_load
ing)
        accept_loading_button.place(x=12, y=126, height=28, width=215)

    def input_validation(self, text, acttyp, entry_type):
        if acttyp == '1':
            if entry_type == "float":
                try:
                    if len(str(int(float(text)))) > 5 or len(str(float(text))) > 15:
                        return False
                except Exception:
                    return False
            else:
                if not text.isdigit() or len(text) > 4:
                    return False
        return True

    def graph_add_loading(self):
        if not self.graph.block:
            f, t, v, r = self.from_node_entry.get(), self.to_node_entry.get(), self.value_inpu
t_entry.get(), \
                self.with_random.get()
            if f and t and v:
                status = self.graph.add_graph_loading(int(f), int(t), float(v), r)
                if status:
                    while self.graph.block:
                        self.canvas.draw()
                        self.main_window.update()
                    if self.graph.add_loading_result:
                        self.cache['last_way_details'] = self.graph.result
                        # self.from_node_entry.delete(0, 'end')
                        # self.to_node_entry.delete(0, 'end')
                        # self.value_input_entry.delete(0, 'end')
                        self.update_edges()
                    else:
                        messagebox.showinfo('System error', 'Node can`t be reached. Find bound
 limit.')
                else:
                    messagebox.showinfo('Data error', 'Node does not exist.')
            else:
                messagebox.showinfo('Data error', 'Fields aren`t filled.')

    def change_display_speed(self, value: int):
        speed_converter = {1: 0.12, 2: 0.08, 3: 0.05, 4: 0.03, 5: 0.015, 6: 0.008}
        self.graph.speed = speed_converter[value]

    def open_statistics(self):
        if self.cache['last_way_details']:
            statistics_window = Toplevel(self.main_window, bg="#636363", name='statistics_wind
ow')
            statistics_window.geometry('620x399+115+115')
            statistics_window.resizable(False, False)
            statistics_window.title("Last way statistics")
            statistics_window.iconbitmap('icons/statistics_ico.ico')

            figure = plt.figure(figsize=(5.84, 3.35))
            grid = gridspec.GridSpec(ncols=1, nrows=2, height_ratios=[2, 2], figure=figure, le
ft=0, right=1, bottom=0.1)

            expected_plot = figure.add_subplot(grid[0])
            realized_plot = figure.add_subplot(grid[1])

            expected = self.cache['last_way_details']['expected']
            realized = self.cache['last_way_details']['result']
```

```
            expected_plot.title.set_text('Expected way, total cost : %s' % round(expected['cos
t'], 3))
            realized_plot.title.set_text('Realized way, total cost : %s' % round(realized['cos
t'], 3))

            split_parts = max([len(expected['path']), len(realized['path'])]) - 1

            def draw(plot, path):
                x, y = [x * (split_parts/(len(path) - 1)) for x in range(len(path))], [0 for _
 in range(len(path))]
                plot.plot(x, y, marker='o', markersize=14, color='#570078', markerfacecolor=(0
.34, 0, 0.47, 0.5))
                for j, dot in enumerate(path):
                    plot.annotate(dot, (x[j], y[j]), fontsize=9, ha='center', va='center')
                plot.axis('off')

            draw(expected_plot, expected['path'])
            draw(realized_plot, realized['path'])

            header = Label(statistics_window, text="Details", width=15, font=("Lato", 14), bg=
"#636363", fg="#000000")
            header.place(x=220, y=7)

            graphic = Frame(statistics_window, width=600, height=350, bg="#262626")
            graphic.place(x=10, y=40)

            canvas = FigureCanvasTkAgg(figure, graphic)
            canvas.draw()
            canvas._tkcanvas.place(x=8, y=8)
        else:
            messagebox.showinfo("Last way information", 'Nothing to show.')
```